Networks · Communications

DECnet–RSX

Programmer's Reference Manual

AA–M098D–TC

digital

DECnet–RSX

Programmer's Reference Manual

Order No. AA–M098D–TC

October 1987

This manual describes the DECnet–RSX programming facilities and provides reference information on network programming calls.

Supersession/Update Information:	This is a new manual.
Operating System and Version:	RSX–11M V4.2 RSX–11S V4.2 RSX–11M–PLUS V4.0 Micro/RSX V4.0
Software Version:	DECnet–11M V4.3 DECnet–11S V4.3 DECnet–11M–PLUS V4.0 DECnet–Micro/RSX V4.0

digital

AA-M098D-TC
First Printing, October 1987

The postage-prepaid Reader's Comments form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	PDP	VAX
DECmate	P/OS	VAXcluster
DECnet	Professional	VAXmate
DECUS	Rainbow	VMS
DECwriter	RSTS	VT
DIBOL	RSX	Work Processor
digital™	RT	
MASSBUS	UNIBUS	

This manual was produced by Networks and Communications Publications.

SW

Contents

2 DECnet–RSX MACRO–11 Programming Facilities

3 FORTRAN, COBOL, and BASIC–PLUS–2 Programming Facilities

4 DLX Ethernet Programming Facilities

5 DLX Point-to-Point and Multipoint Programming Facilities

C Remote File Access Error/Completion Codes

D MACRO–11 Connect Block Offset and Code Definitions

E Network Error/Completion Codes for FORTRAN, COBOL, and BASIC–PLUS–2

F Network MACRO–11 Error/Completion Codes

G Values for Ethernet and 802.3 Addressing

H DLX Characteristics Status Codes

Figures

Tables

Preface

The *DECnet–RSX Programmer's Reference Manual* explains DECnet programming concepts and describes the DECnet–RSX calls for the following programming functions:

- Intertask communication
- Remote file access
- Task control
- Direct line access (DLX)

The DECnet–RSX software supports intertask communication calls for MACRO–11, FORTRAN 77, COBOL, and BASIC–PLUS–2 programming; remote file access calls for FORTRAN 77, COBOL, and BASIC–PLUS–2; task control calls for FORTRAN 77; and QIO calls for the DLX user interface.

The manual also includes information on writing applications for Local Area Transport (LAT) application terminals.

Throughout the manual, the term "DECnet–RSX" refers to all the software that you receive in your DECnet–RSX distribution kit.

Intended Audience

This manual is for users who write network programs that run on DECnet–11M, DECnet–11M–PLUS, DECnet–11S, and DECnet–Micro/RSX systems.

Structure of This Manual

The manual is organized as follows:

Chapter 1	Provides introductory information about intertask communication, remote file access, and task control operations.
Chapter 2	Describes the DECnet–RSX MACRO–11 programming facilities for intertask communication macros.
Chapter 3	Describes the DECnet–RSX FORTRAN, COBOL, and BASIC–PLUS–2 programming facilities for intertask communication, remote file access, and FORTRAN task control.
Chapter 4	Describes the DECnet–RSX Direct Line Access (DLX) programming facilities and QIO calls for tasks using the Ethernet.
Chapter 5	Describes the DECnet–RSX Direct Line Access (DLX) programming facilities and QIO calls for tasks using point-to-point and multipoint lines.
Chapter 6	Describes programming facilities for tasks that use Local Area Transport (LAT) application terminals.
Appendix A	Contains the network disconnect or reject reason codes.
Appendix B	Defines the Digital object type code values.
Appendix C	Summarizes remote file access error/completion codes.
Appendix D	Contains the MACRO–11 connect block offset and code definitions.
Appendix E	Contains the FORTRAN, COBOL, and BASIC–PLUS–2 network error/completion codes.
Appendix F	Contains the MACRO–11 network error/completion codes.
Appendix G	Provides information on multicast addresses, protocol types, and protocol identifiers to use with Ethernet applications.
Appendix H	Describes DLX completion codes for characteristics operations in Ethernet applications.

Associated Documents

The following manuals are part of the DECnet–RSX documentation set:

- *DECnet–RSX Network Management Concepts and Procedures*
- *DECnet–RSX Guide to Network Management Utilities*
- *DECnet–RSX Guide to User Utilities*
- *DECnet–RSX Network Generation and Installation Guide*
- *DECnet–RSX User's Pocket Guide*
- *DECnet–RSX Programmer's Pocket Guide*
- *DECnet–RSX Network Manager's Pocket Guide*
- *DECnet–RSX Release Notes*

The RSX–11M documentation set and the appropriate programming language manuals are also helpful.

For information on LAT, refer to the *Local Area Transport (LAT) Network Concepts* manual.

For information on how to use IEEE 802.3 frame format for DLX programs, refer to the following two standards:

- *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, ANSI/IEEE Std 802.3–1985, ISO Draft International Standard 8802/3.
- *Logical Link Control*, ANSI/IEEE Std 802.2–1985, ISO Draft International Standard 8802/2.

These standards are published by the Institute of Electrical and Electronics Engineers, Inc., and distributed in cooperation with Wiley-Interscience, a division of John Wiley & Sons, Inc.

Acronyms

The following acronyms are used in this manual:

AST	Asynchronous system trap
CEX	Communications Executive
DLX	Direct Line Access Controller
FCS–11	File Control System
MOP	Maintenance Operation Protocol
NETFOR.OLB	DECnet high-level language library
NETLIB.MLB	DECnet MACRO–11 library
NFAR	Network File Access Routine
PSW	Processor status word
QIO	Queued input/output call

Graphic Conventions

Convention	Meaning
`Special type`	This special type shows examples of user input (in red) or system output (in black).
UPPERCASE	Uppercase letters indicate characters to type exactly as shown. You can type the text in upper- or lowercase. You can abbreviate uppercase words to the first three or more unique characters.
lowercase italics	Lowercase italics indicate variables for which you specify or the system supplies the actual values.
[ ]	Square brackets enclose optional data. If the brackets enclose a vertical list of options, you can specify only one option. Do not type the brackets when you enter the call.
{ }	Braces enclose options, from which you must choose one and only one. Do not type the braces when you enter the call.
( )	Parentheses enclose a set of options. You must specify both or neither of them. Do not type the parentheses when you enter the call.

Convention	Meaning
(*key*)	This symbol indicates a key to press. (CTRL/*x*) indicates that you press the CONTROL key and the key represented by *x* together.

`PRST2=5`
`.`
`.`
`.`
`CALL BFMT1`
	Ellipses represent an omission. To emphasize the important information, examples may omit some of the user input or system output

All values that appear in this manual are decimal unless otherwise noted.

Certain additional conventions apply in specific chapters. Section 2.4 describes specific conventions for MACRO–11 calls. Section 3.7 describes specific conventions for FORTRAN, COBOL, and BASIC–PLUS–2 calls.

1
Introduction

DECnet–RSX software extends the RSX operating systems for the PDP–11. With DECnet–RSX, you can write programs that exchange data with programs on other DECnet systems in the network that run under RSX or other operating systems. This manual describes the network programming functions that you can use with the MACRO–11, FORTRAN 77, COBOL, and BASIC–PLUS–2 languages.

Programs using DECnet–RSX can have tasks running on different nodes that exchange data using intertask communication, remote file access, or remote task control.

- **Intertask communication.** User tasks on different nodes can exchange messages and data by issuing a series of DECnet communications calls. DECnet software lets you perform task-to-task communication, whether or not both tasks are written in the same programming language or both nodes are under the same operating system. For example, an RSX FORTRAN 77 program can communicate with a VMS BASIC–PLUS–2 program.

- **Remote file access.** Remote file access programs can access sequential files for reading, writing, and appending records to remote files. They can also delete remote files.

  Programming remote file access operations on a remote RSX node is similar to programming local I/O operations. For a remote node under another operating system, your program's access is determined by what functions its source language provides and the remote file system supports.

- **Remote task control.** A FORTRAN program can control the execution of installed tasks on remote RSX or IAS DECnet nodes. The program can cause immediate execution of a remote task, schedule it for later or periodic execution, abort it, or cancel its scheduling.

You can perform functions on cooperating local and remote nodes or simply at the local node. For example, a task can issue MACRO–11 DECnet calls to exchange data with another task on the same node. This lets you debug an RSX program locally before running it at a remote node.

1.1 Intertask Communication Conventions

To refer to a call common to all four languages that DECnet–RSX supports, this manual uses the call's first three letters, followed by *x*. The *x* represents the part of the call name that varies by programming language. With MACRO–11, you replace the *x* with the $ symbol. With FORTRAN, COBOL, and BASIC–PLUS–2, you replace the *x* with the letters NT.

In this manual, the term "macro" refers to MACRO–11 calls; the term "call" refers to FORTRAN, COBOL, and BASIC–PLUS–2 calls.

Example:

OPN*x* (generic representation)
OPN$ (MACRO–11)
OPNNT (FORTRAN, COBOL, or BASIC–PLUS–2)

OPN*x* is the first DECnet call a task issues for any DECnet session. A session includes all intertask communication calls issued between the OPN*x* call and the CLS*x* call.

Table 1–1, in Section 1.3, provides an alphabetical list of the DECnet intertask communication calls, and describes the function and expected result of each.

1.2 Intertask Communication Concepts

This section explains the following basic intertask communication concepts:

- Establishing an active network task
- Establishing a logical link
- Building a connect block
- Getting data from the network data queue
- Sending and receiving messages

- Sending interrupt messages
- Checking completion status information
- Terminating activity on a logical link
- Closing a network connection

1.2.1 Establishing an Active Network Task

Before any task can exchange data using intertask communication calls, it must be an active network task. A task is active if it is running and has issued an open (OPN*x*) call. An OPN*x* call connects the task to the network. It can also establish the task's network data queue.

Your task requires Logical Unit Numbers (LUNs) assigned to NS:, the network pseudodevice driver: a network data queue (mailbox) LUN and a LUN for each logical link.

The network data queue (mailbox) LUN specifies the queue from which your task retrieves network messages.

You can define the network data queue LUN by including the value as an argument to the OPN$/OPNNT call or you can use the value of the symbol .MBXLU. You can define the value of .MBXLU at task build time; with MACRO–11, you can also define it at assembly time. The .MBXLU definition is referenced only if you omit the LUN argument in the call; so define a particular network data queue LUN in only one place.

To define .MBXLU at task build time, issue the following task build option:

```
GBLDEF=.MBXLU:x
```

This instructs the task builder to define all global references to .MBXLU as the integer value represented by *x*.

Using MACRO–11, you can choose to define .MBXLU at assembly time. To define the LUN locally in your source code to a value represented by *x*, put the following in each source module:

```
.MBXLU=x
```

To define the LUN globally, include this statement in a single source module:

```
.MBXLU==x
```

The task builder will define references to .MBXLU in each source module in your task to the value of *x*.

A task can use only one network data queue LUN. In MACRO–11 tasks, you can specify a network data queue LUN with the following calls: SPA$ (specify a user AST routine), GND$ (get network data), REJ$ (reject a logical link request), CLS$ (end the task's network operations), and GLN$ (get local node information). You must specify the same LUN that you specified in the OPN$ call, or the macro completes with a privilege error (IE.PRI).

Choose only one of these command techniques for defining .MBXLU. However, if you do not define .MBXLU, the task builder returns an undefined reference warning message. If you run the task and ignore the warning, the operating system rejects the macro calls (OPN*x*, SPA$, GND$, REJ$, CLS$, and GLN$) with a directive status error indicating an invalid LUN. The task builder causes undefined references to default to zero (0). You cannot define .MBXLU with a value of 0; 0 is an invalid logical unit number.

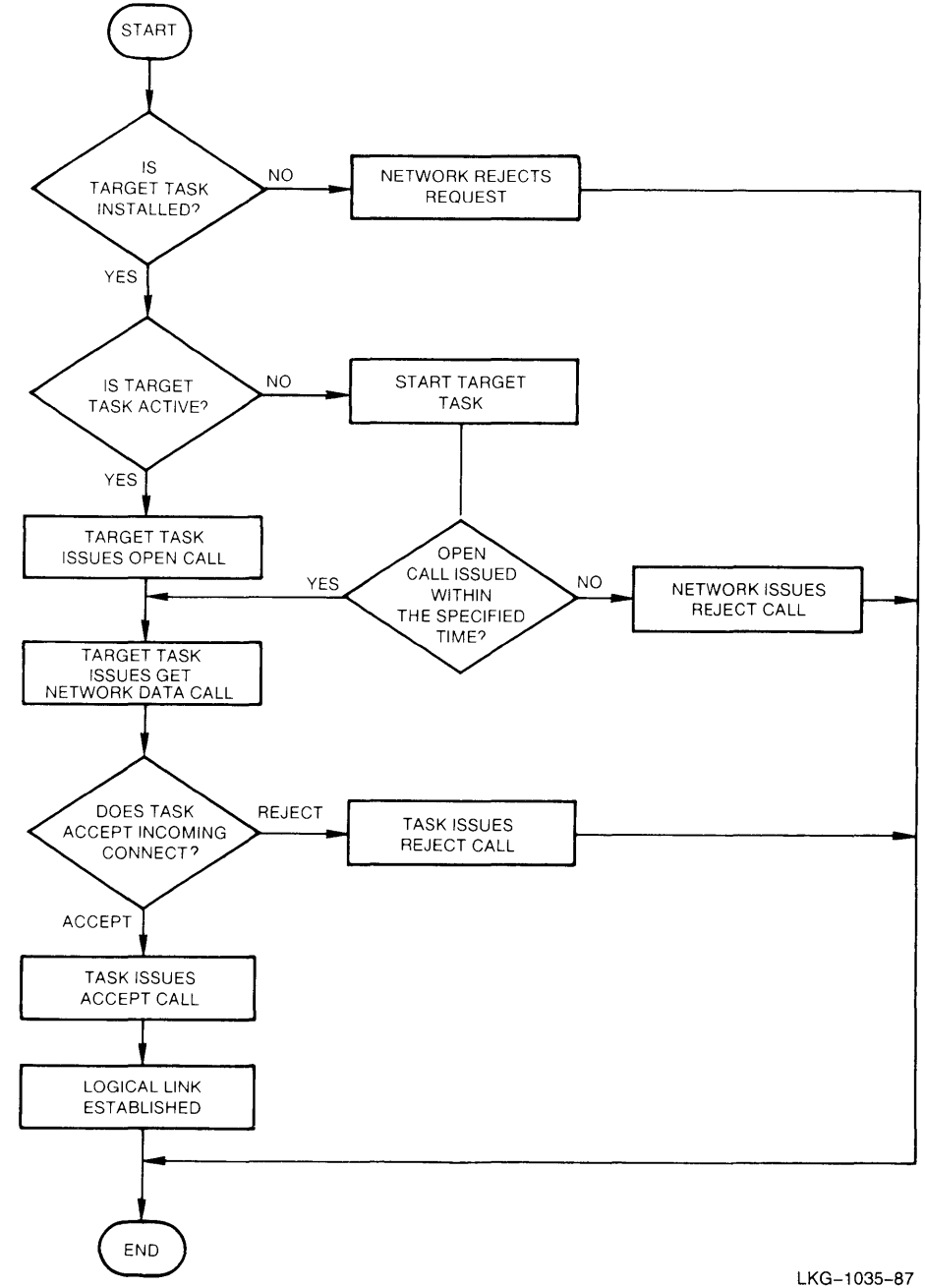
Issuing a CLS$ call to terminate network operations for a MACRO–11 task frees all network logical unit numbers and aborts the task's logical links. You can issue the CLS$ call in any of the three CLS$ formats – CLS[W]$, CLS[W]$E, and CLS[W]$S.

If a LUN is not assigned to NS:, any network directive is returned with the "illegal function code" status.

1.2.2 Establishing a Logical Link

To exchange data, a logical link must exist between two active network tasks. A logical link is a logical path between two cooperating tasks that agree to communicate. When the link is established, a user task can send and receive messages. Figure 1–1 illustrates the flowchart process for establishing a logical link.

Figure 1–1: Establishing a Logical Link



LKG–1035–87

The task requesting the logical link is the source task. The task receiving the request is the target task. The distinction between source and target task applies only during the connection sequence. Once the logical link is established, both tasks have equal access to it.

Tasks at both ends of the link must specify a logical unit number (LUN) for the link. Each task assigns a link LUN to identify the link to the tasks and network. The tasks at both ends need not use the same LUN for a link.

1.2.3 Building a Connect Block

Before the source task can issue a request to connect to another task, it must build a connect block. A connect block contains a destination descriptor, source descriptor, and optionally, access control information and user-supplied data.

1.2.3.1 Destination Descriptor

The destination descriptor identifies the destination task by task name or object type number.

When two tasks communicate, they are considered to be two objects. A task can be set up as a named or numbered object:

- A named object is an installed user-defined task to which you connect by specifying a name. The object type number for a named object is 0.

- A numbered object is an installed user-defined task or DECnet task to which you connect by specifying an object type number. The object type numbers for numbered objects range from 1 to 255, with 1 to 127 reserved for DECnet tasks and 128 to 255 reserved for user tasks.

For information on defining objects, see the *DECnet–RSX Network Management Concepts and Procedures* manual.

1.2.3.2 Source Descriptor

The source descriptor contains information that the DECnet software on the source node supplies. The information includes either

- The source node name and task name of a named object or the source node name and task object number of a numbered object.

- The source node name and the user name of the person running the task.

The target task can use the source descriptor to determine whether it wants to establish communications.

1.2.3.3 Access Control Information

Access control information defines your access rights at the remote node. Each target system performs access control verification according to its conventions. If the target node is equipped to verify access control information, it does so before passing the connect request to the target task. For information on access control verification, see the *DECnet–RSX Network Management Concepts and Procedures* manual.

1.2.3.4 Optional Data Messages

When the source task issues a connect request, you can include a data message of up to 16 characters in the connect block. If the connect (CON*x*) call contains the location and length of a block of user data, the source node appends that block to the connect block.

1.2.4 Getting Data from the Network Data Queue

Once a task is connected to the network, it has a network data queue. The software on the connected task's node places all incoming connect request messages, interrupt messages, user disconnect messages, user abort messages, and network abort messages on the task's network data queue. To get these messages, the task issues a get network data (GND*x*) call. A task should begin monitoring its network data queue as soon as the open call completes successfully.

The get network data call ordinarily returns the first message on the queue on a first-in, first-out basis. However, the GND*x* call has the following options:

- Remove the first message on the queue and place it in the message buffer.
- Remove the first message of a specified type for any logical link and place it in the message buffer.
- Remove the first message for a specified logical link regardless of the message type and place it in the message buffer.
- Remove the first message of a specified type for a specified logical link and place it in the message buffer.
- Determine the type, length, and associated logical link of any message on the queue without removing it from the queue. This allows you to assign an appropriate buffer size in a subsequent GND*x* call that performs one of the four options just listed.

1.2.5 Sending and Receiving Messages

Once a logical link is established between two tasks, both tasks can send and receive messages. DECnet distinguishes between data and nondata messages. Data messages go directly to a buffer provided by the receiving task. Nondata messages go to a task's network data queue. Nondata messages are unsolicited high priority messages that inform the receiving task of an event, such as an interrupt or disconnect request.

To send a data message, a task issues a send (SND*x*) call. The send call must specify the LUN that the connect or accept call assigned. It also specifies the location and length of the data message buffer. A send call completes when the receiving node acknowledges to the sending node that it received a message correctly.

To receive a data message, a task issues a receive (REC*x*) call. The receive call must specify the LUN that the connect or accept call assigned. It also specifies the location and length of the data message buffer. A receive call completes when the data message is placed in the specified data message buffer. If the data message buffer is not large enough, the receive call completes with a data overrun condition and the excess data is lost; the I/O status indicates the overrun in such cases. Another receive call is then required to receive the next data message.

To send a high priority nondata message, a task issues an abort (ABT*x*), disconnect (DSC*x*), or interrupt (XMI*x*) call. To receive a high priority nondata message, a task issues a get network data (GND*x*) call.

1.2.6 Sending Interrupt Messages

A task can send interrupt messages to another task. Usually an interrupt message informs the receiving task of some unusual event in the sending task. An interrupt (XMI*x*) message can be up to 16 bytes long. In the interrupt call, specify the LUN assigned in the connect or accept call. Also specify the location and length of the message buffer.

An interrupt call completes when the receiving node acknowledges to the sending node that it received the message. The receiving node software places the interrupt message on the receiving task's network data queue. The receiving task must issue a get network data call to remove the message from the queue and place it in the task's message buffer.

A task can have only one interrupt message outstanding on a logical link. Until the call completes, any subsequent attempt to send another interrupt message on that same link will return an error code in the I/O status word.

1.2.7 Checking Completion Status Information

Each macro or call can include an argument that specifies the address of a 2-word status block. The status block returns the status of a completed call. Include this argument so that you can check completion status. You can use the same block for successive I/O requests, but only by one macro or call at a time. If concurrent I/O requests attempt to use the same status block, unpredictable results occur.

The first status word contains:

- A zero if the called macro or subroutine has not completed
- A positive value if the called macro or subroutine produced the desired results
- A negative value if the called macro or subroutine did not produce the desired results

The second status word contains further information about the completion. For example, in a successful data transmission, it returns the number of bytes transmitted.

1.2.8 Terminating Activity on a Logical Link

Any task can terminate activity on a logical link at any time by issuing a disconnect or abort call:

- A disconnect (DSC*x*) call terminates transmissions over the logical link after all data transmissions and interrupts have been sent.
- An abort (ABT*x*) call disconnects the logical link immediately, even if messages are queued for transmission.

The receiving node software places the termination message on the receiving task's network data queue. The receiving task must issue a get network data call to retrieve the message.

Both disconnect and abort calls can specify the location and length of a user data message of up to 16 bytes for the receiving task.

The disconnect call must specify the logical unit number (LUN) assigned in the connect or accept call. A disconnect call allows node software to complete all pending transmits for the issuing task before disconnecting the logical link. During this time, the issuing task continues to receive messages. When the last message is transmitted, however, any remaining receive calls complete with an abort

condition. When the link is disconnected, the LUN is freed. A task can use that LUN in subsequent connect or accept calls.

The abort call must specify the logical unit number (LUN) assigned in the connect or accept call. An abort call causes the node software to immediately abort all pending transmits and receives and disconnect the link. The LUN is freed and a task can use that LUN in subsequent connect or accept calls.

1.2.9 Closing a Network Connection

To close a task's network connection, issue a close (CLS*x*) call. The close call informs the node software that the task no longer requires network services and purges the task's network data queue. Any active LUNs are deactivated and freed for use if the task subsequently issues an open (OPN*x*) call.

If data remains in the terminating task's network data queue when the close call is issued, the task receives any connect requests that remain in its network data queue if it subsequently issues an open call within a short period of time. Data of other types, such as interrupt, disconnect, and abort messages, are discarded.

1.2.10 Using the Wait Option

Many macros and calls allow you to use the wait option. Including W in a call (such as GNDW$ or GNDNTW) delays execution of the calling task until the call completes. The calling task then continues at the instruction immediately following the call. Without the wait option, the call executes asynchronously.

In a MACRO–11 call for which you specify the wait option, an event flag is mandatory. If you omit the event flag, the call completes as a normal asynchronous call.

1.2.11 Using the AST and WAITNT Options

An asynchronous system trap (AST) in a MACRO–11 call causes the AST to execute when the call completes. In FORTRAN, COBOL, and BASIC–PLUS–2, the WAITNT call instead determines when a call completes.

1.2.12 Using the Flow Control Option

A network program requires buffer space for temporary message storage. For example, a program keeps a copy of each message that it sends over the link in buffer space until the receiver acknowledges the message. A program also holds buffer space for receiving inbound messages.

DECnet provides flow control mechanisms that prevent the overflow of available buffer space. Sending and receiving tasks are synchronized so that a source task transmits data only if the target task has issued a receive call and has available buffer space.

With MACRO–11 tasks, DECnet–RSX also provides a special NOFLOW option that disables flow control mechanisms. The NOFLOW option can help in attaining a higher level of network performance, but you must use it with caution. Without flow control, a source task can send data whether or not a buffer is available to receive it. If the target task does not have adequate buffering for the incoming data, some data segments will be discarded. The software must request retransmission of each discarded segment, after a timeout. This significantly degrades network performance. If you choose the NOFLOW option, maintain adequate buffering at the target task to compensate for the loss of send/receive synchronization. The communicating programs should be appropriately written.

The NOFLOW option is desirable when:

- A program's User layer protocol already includes control mechanisms or acknowledgment-signaling mechanisms.
- The flow of data is predictable, and the program can handle the flow.

NOTE

It is inadvisable to use the NOFLOW option in the initial stages of developing a network program. Wait until after you test the communicating programs and adequately synchronize the data flow.

You can set either end of the logical link to FLOW or to NOFLOW control independently. Set the NOFLOW control option within the CON*x* and ACC*x* calls. Flow control is the default.

1.3 Summary of Intertask Communication Calls

Table 1–1 lists the intertask communication calls. The first column of the table lists the call name. MACRO–11 macro call names have the form *nam*$ and FORTRAN, COBOL, and BASIC–PLUS–2 call names have the form *nam*NT, where *nam* represents the specific call. The second column defines the function of the call. The third column describes the result of the call's successful execution.

Table 1–1: DECnet Communication Calls

Call	Function	Normal Action
ABT$ ABTNT	Abort a logical link	Transmits a nondata abort message over the logical link. The DECnet software on the receiving node delivers the abort message to the receiving task's network data queue.
ACC$ ACCNT	Accept a logical link request	Notifies the DECnet software on the target node that the target task accepts a logical link request. The DECnet software then sends the acceptance notification to DECnet software on the source node. The source node DECnet software delivers the accept message to the status block that the source task specified in the CON$ or CONNT call.
BACC BACCL	Build access control information area	Builds the access control information for a connect block. Use BACC for a short connect block, and BACCL for a long connect block. A subsequent CONNT call delivers the contents of this block to the DECnet software on the target node. This call does not transmit user data over a logical link.
BFMT0 BFMT1	Build a format descriptor block	Builds a destination descriptor for the connect block. A format 0 destination descriptor describes the target task by object code. A format 1 destination descriptor describes the target task by name. This call does not transmit user data over a logical link. A subsequent CONNT call delivers the contents of this block to the DECnet software on the target node.
CLS$ CLSNT	Close the network connection — end the task's network operations	Sends the close request to the DECnet software on the issuing task's node. This call does not transmit user data over a logical link.
CON$ CONNT	Request a logical link connection	Sends a high priority nondata connect message and connect block over a temporary logical link to the target node's DECnet software. If the target task is an active network task, the connect request is delivered to its network data queue. The connect block is delivered to a mailbox as a result of a subsequent get network data call (GND$ or GNDNT).

Table 1–1 (Cont.): DECnet Communication Calls

Call	Function	Normal Action
CONB$$	Build a connect block (short)	Builds a connect block that a subsequent CON$ call delivers to the DECnet software on the target task. This connect block accepts a user ID of up to 16., password of up to 8., and account number of up to 16. bytes. This call does not transmit data.
CONL$$	Build a connect block (long)	Builds a connect block that a subsequent CON$ call delivers to the DECnet software on the target task. This connect block accepts a user ID, password, and account number of up to 39. bytes each. This call does not transmit data.
DSC$ DSCNT	Disconnect the logical link	Transmits a nondata disconnect message over the logical link. The DECnet software on the receiving task's node delivers the disconnect message to the task's network data queue.
GLN$ GLNNT	Get local node data: node name and transmission segment size	Delivers local node information to the buffer that an argument of the call specifies. This call does not transmit user data over a logical link.
GND$ GNDNT	Get network data from tasks's network data queue	Stores data from the network data queue in the location that arguments of the call in the issuing task specify. This call does not transmit user data over a logical link.
OPN$ OPNNT	Open the network connection — create the task's network data queue	Delivers the open request to the DECnet software on the issuing task node. This call does not transmit user data over a logical link.
REC$ RECNT	Request to receive data over the logical link	Receives a data message that another task's DECnet send call initiated. When the receive call completes, the DECnet software on the issuing task node sends a notification message to the status block specified in the call. This call does not transmit user data over a logical link.
REJ$ REJNT	Reject a logical link request	Sends rejection notification over the temporary logical link from the DECnet software on the target task's node to the DECnet software on the source node. The source node's DECnet software delivers the reject notice to the status block that the source task's CON$ or CONNT call specified.

(continued on next page)

Table 1–1 (Cont.): DECnet Communication Calls

Call	Function	Normal Action
SND$ SNDNT	Request to send a data message over the logical link	Transmits the data message over the logical link to the DECnet software on the receiving task node. When the receiving DECnet software delivers the message to the area specified in the receiving task's receive call, it returns a completion status message to the sending DECnet software, which delivers it to the status block specified in the send call.
SPA$	Specify the location of a user-written asynchronous system trap (AST) routine	Transfers control to the AST routine when a nondata message is placed on the task's network data queue. This call does not transmit user data over a logical link.
WAITNT	Wait for the completion of any other DECnet communications call	Suspends task execution until completion of a previously-issued call that included a wait option. This call does not transmit user data over a logical link.
XMI$ XMINT	Request to send an interrupt message over the logical link	Transmits a high priority nondata interrupt message over the logical link. The DECnet software on the receiving node delivers the interrupt message to the receiving task's network data queue.

1.4 DECnet–RSX Remote File Access Operations

Using DECnet–RSX remote file access facilities, you can write a FORTRAN, COBOL, or BASIC–PLUS–2 program that performs the following file access operations for sequential files only:

- Open or create a remote file
- Read and write records to a remote file
- Append records to a remote file
- Close, purge, or delete a remote file

DECnet–RSX file access facilities have similar features to those of DECnet–RSX intertask communication facilities.

- The file access facilities are implemented by means of calls to subroutines.
- The task that requests file access is called the source task, and the task that accepts or rejects the request is called the target task.
- Acceptance of a file access request creates a logical link between the source and target tasks. Then the file access process begins.

Incoming file access requests are translated into calls to the file system at the target node. The resulting file data is sent back to the accessing task. The accessing task then reformats the data as the system requires. The DECnet software establishes the logical link for file access operations; much of the connection process is therefore transparent, in contrast to intertask communication. After completing file access operations, the logical link is disconnected.

Section 3.9 discusses remote file access operations.

1.5 DECnet–RSX Task Control

DECnet–RSX task control lets you write tasks in FORTRAN that:

- Execute an installed task on a remote node according to a set schedule, using the RUNNCW call
- Abort an executing task on a remote node, using the ABONCW call
- Cancel a scheduled task on a remote node, using the ABONCW call

Section 3.10 discusses remote task control.

2

DECnet–RSX MACRO–11 Programming Facilities

DECnet–RSX provides a library of MACRO–11 macros to use in network intertask communication. This chapter:

- Describes the three network macro formats.
- Explains the connect block and access options for your task.
- Lists the graphic conventions for this chapter.
- Describes each intertask communication macro call.

2.1 RSX–11 Network Macro Formats

You can use the following formats to code macros:

- **BUILD type macro.** Creates a parameter block at assembly time and is generally used in conjunction with an EXECUTE type macro or a DIR$ directive.
- **EXECUTE type macro.** References the parameter block that a BUILD type macro created and executes the requested function. An EXECUTE type macro lets you override parameters that the BUILD type macro specified.
- **STACK type macro.** Creates a parameter block on the processor stack and executes the requested function.

Section 2.1.4 has examples of these three macro format types.

2.1.1 BUILD Type Macros

You use the BUILD type macro at assembly time. This macro creates a parameter block that contains arguments that describe the network function you requested. A pre-defined parameter block is especially useful for repetitions of the same network operation. If you omit an optional parameter from this block, the macro allocates space for it anyway. Later, you can use an EXECUTE type macro to fill in the argument. If you plan to use the EXECUTE type macro, however, you must include all of the BUILD type macro's trailing arguments.

The format for a BUILD type macro is:

label: *xxx*[W]$ *parameter-list*[,*flag*]

where

label — is a symbolic name associated with the location of the parameter block.

xxx — is the name of a DECnet–RSX macro.

[W] — specifies that this network function will complete synchronously. The issuing task waits until the function completes before continuing. If you omit the W, the call completes asynchronously.

parameter-list — is a list of arguments that describe particular features of this call. Each parameter must be a valid argument for a .WORD or .BYTE MACRO–11 directive. Each call description includes a list of the parameters for the call. The number of arguments specified must not exceed the number specified in an EXECUTE type macro that will use this parameter block.

flag — is a symbolic name that specifies an optional subfunction of the network macro.

You can execute a BUILD type macro by issuing an EXECUTE type macro or a DIR$ macro. A DIR$ macro call pushes the address of the network function parameter block that you created with the BUILD macro on the processor stack and then issues an EMT 377 for the Executive to execute the macro. You can write a DIR$ macro as follows:

DIR$ *adr,err*

where

adr is the address of the parameter block in the format of a source operand of an MOV instruction.

err is the address of an optional error routine. The C-bit in the processor status word (PSW) is set whenever an error is encountered.

A DIR$ macro generates less code than a corresponding EXECUTE macro.

2.1.2 EXECUTE Type Macros

The EXECUTE type macro references a network function parameter block that you create at assembly time with a BUILD type macro. An EXECUTE type macro lets you enter parameters that override those that you originally defined with a BUILD macro.

You must include all trailing arguments in a BUILD type macro that an EXECUTE type macro references.

Once you redefine or specify new parameters for the call, the EXECUTE macro automatically executes the function that the call requests. The format for an EXECUTE type macro is:

xxx[W]$E *label*[,*override-parameter-list*][,*flag*]

where

xxx is the name of a DECnet–RSX macro.

[W] specifies that this network function will complete synchronously. The issuing task waits until the function completes before continuing. If you omit the W, the call completes asynchronously.

label represents one of two values:

- The *label* of the BUILD type macro that supplies parameters for this EXECUTE macro. You can include arguments immediately following the *label* to override any parameters previously defined in the BUILD parameter block.

- The *label* of an area of memory that will contain the parameters that you are currently specifying. The parameter block is built and the call is executed in the same macro.

override-parameter-list is a list of one or more arguments to replace parameters that you previously defined for this call in a BUILD type macro. Each argument in this list must be a valid source operand for an MOV S, label + offset MACRO–11 instruction.

You can override the value of a parameter and assign it a null value. For example, if an AST is not required, specify the parameter as 0.

flag is a symbolic name that specifies an optional subfunction of the network macro.

2.1.3 STACK Type Macros

The STACK type macro creates the network function parameter block for the call on the processor stack and then executes the requested function. You must specify all required parameters when you issue this macro or it will generate assembly errors.

The format for a STACK type macro is:

xxx[W]$S *parameter-list*[,*flag*]

where

xxx is the name of a DECnet–RSX macro.

[W] specifies that this network function will complete synchronously. The issuing task waits until the function completes before continuing. If you omit the W, the call completes asynchronously.

parameter-list is a list of arguments that describe particular features of this call. Each argument must have the form of a valid MACRO–11 MOV instruction source operand. Each call description includes a list of the parameters for the call.

flag is a symbolic name that specifies an optional subfunction of the network macro.

2.1.4 Macro Format Examples

The following examples demonstrate the three macro types. The first, a BUILD type macro, creates a parameter block for the call designated by *xxx*$:

label: *xxx*[W]$ *lun,efn,status,ast,<p1,p2,...,pn>*

The first example of an EXECUTE type macro references the parameter block created for *label*:

xxx[W]$E *label*

The second EXECUTE type macro overrides the parameter list arguments *p1* and *p2*:

xxx[W]$E *label,,,,,<p1,p2>*

The last example, a STACK type macro, creates a parameter block on the stack, and executes the call.

xxx[W]$S *#lun,#efn,#status,#ast,<#p1,#p2>*

2.2 Connect Block Options

As Chapter 1 described, a source task builds a connect block before issuing a connect request. This outgoing connect block contains information about the connect request's target node and task. It can also specify explicit access control information that gives the source task access to the target node. Before network software sends the connect block to the target task, it adds information about the source task or user. If you have an RSX–11M–PLUS or Micro/RSX system with outgoing proxy enabled, network software also adds proxy information (see Section 2.3). At the target node, the target task retrieves the incoming connect block from the network data queue.

Your task can use either long or short connect blocks. Using long connect blocks lets your task support user IDs, passwords, and accounts of 39. characters each. Using short connect blocks lets your task support user IDs of up to 16. characters, passwords of up to 8. characters, and accounts of up to 16. characters.

For greatest flexibility, use long connect blocks when writing a new task. However, you can continue to use an existing task that uses short connect blocks without modifying the task. If you change an existing task to use long connect blocks, note the added buffer space requirements. Also note that if the task uses proxy access, you need not supply values for the access control information fields.

2.2.1 Using Connect Block Options

The connect block size that you choose affects the following macro calls in your task:

Macro	Connect Block Option
OPN$	Include the NT.LCB flag to specify a long connect block.
CONB$$	Use CONB$$ to build a short connect block.
CONL$$	Use CONL$$ to build a long connect block.
CON$	Provide the appropriate connect block length in the *conblen* argument.
ACC$ GND$ REJ$	Use the *mail* and *mailen* arguments to reference the appropriately-sized buffer.

When access verification for your task is on, your node's network software verifies access rights and removes the access control information before passing an incoming connect block to your task. For information on enabling verification for a task, refer to the *DECnet–RSX Guide to Network Management Utilities*.

2.2.2 Receiving Connect Block Information

You specify the type of connect block you want to receive by including or omitting the long connect block (NT.LCB) flag in the OPN$ (access the network) macro. If you specify the NT.LCB flag, network software uses long connect block fields when passing access control information to your task.

In the GND$ macro, which retrieves the connect block from the network data queue, you specify a buffer to hold the incoming connect block information. The buffer size that you allocate may or may not equal the size of the incoming connect block, but in writing incoming data to your buffer, network software always uses the offsets appropriate to the connect block size that your OPN$ macro specified. You receive all information if the source task sends the same size connect block that you receive, or if you receive long connect blocks and the source task sends a short connect block. However, if you receive short connect blocks and the source task sends a long connect block, you may lose some information. Network software writes the received information into the appropriate field if the information fits. Information that does not fit into the receiving field causes a data overrun error and is lost.

You can choose to allocate a receiving buffer that is smaller or larger than the expected connect block. For example, you might allocate a smaller buffer to exclude all but the initial fields, or allocate a larger buffer to receive optional user data. The GND$ call description describes what happens when the task receives access control information that is smaller or larger than expected.

2.3 Access Control Information

An outgoing connect request sends information to the target node in order to gain access to an account on the target node. You can specify the access control information and/or the network software can supply proxy information. Proxy access is available only with RSX-11M-PLUS or Micro/RSX.

When you supply explicit access control information for the connect request, you specify a user ID, password, and, optionally, an account number. These identify the target account on the remote node. You specify the explicit access control information as arguments to the macro that builds the connect block (CONB$$ or CONL$$). When the target system receives the connect request, it grants access according to what you specified. For information on which access control arguments the target system requires, refer to user documentation for that system; DECnet-RSX nodes require the user ID and password. For more information on explicit access control information, refer to the *DECnet-RSX Network Management Concepts and Procedures* manual.

Proxy access, in contrast, eliminates the need to send passwords across the network. The network managers on both nodes must set up the environment for using proxy. Once your network manager enables outgoing proxy, your node automatically sends proxy information with all outgoing connect requests. Proxy information is the user ID under which the source task is executing. If incoming proxy is enabled on the target node, the system can grant access according to the proxy information and source node name. For information on how a target system verifies proxy access, refer to the *DECnet-RSX Guide to Network Management Utilities*.

If an incoming connect request contains both explicit and proxy access control information, the target system uses the explicit information, and not the proxy information, to verify access.

2.4 Conventions Used in This Chapter

The following conventions are used in the macro descriptions and examples in this chapter:

asterisk * — flags arguments that you must check for information after the macro completes. For example, the *status* argument specifies an array/data item where completion status information is stored when the macro completes.

UPPERCASE — indicates characters to type exactly as shown. You can type the text in upper- or lowercase.

lowercase italic — indicate variables for which you specify or the system supplies the actual values.

square brackets [ ] — enclose optional data. If the brackets enclose a vertical list of options, you can specify only one option. Do not type the brackets when you code a macro.

Example:

ABT[W]$ *lun*,[*efn*],[*status*],[*ast*][,<*out*,*outlen*>]

In this macro, the *lun* argument is required; all other arguments are optional.

braces { } — enclose options, from which you must choose one and only one. Do not type the braces when you code the macro.

Example:

GND[W]$ *lun*,[*efn*],[*status*],[*ast*],

$$\left\{\begin{array}{l} <mail,mlen> \\ <mail,mlen,mask>,\text{NT.TYP} \\ ,\text{NT.LON} \\ <,,mask>,\text{NTLON} \end{array}\right\}$$

In this example, you must include one of the four argument strings enclosed within the braces when you code GND$.

commas and angle brackets < >

must be typed as part of the macro format. Even if you omit an argument, include the comma that delineates its field unless no other arguments follow.

Example:

Basic format:

ABT[W]$ *lun*,[*efn*],[*status*],[*ast*][,<*out*,*outlen*>]

Sample macro:

```
ABT$ 5,,status
```

efn, *ast*, *out*, and *outlen* have been omitted. A comma delineates the field for the missing *efn* argument; no commas are necessary for the three arguments dropped at the end of the macro.

numbers

are octal unless followed by a decimal point. If the assembler default radix has been set to octal, you can designate a decimal radix by placing a decimal point immediately after a number.

Example:

```
SND$S #3,#1,#IOSTN,,<#MAUX,#16.>
```

In this example, 16 is a decimal number.

2.5 Intertask Communication Macros

This section contains descriptions and usage guidelines for the intertask communication calls that Table 2–1 lists. Read the preceding material in this chapter before using the calls. If you are unfamiliar with intertask communication concepts, also read Chapter 1 carefully.

Table 2–1: Intertask Communication Macros

Macro	Function
ABT$	Abort a logical link
ACC$	Accept a logical link connect requrest
CLS$	End a task's network operations
CON$	Request a logical link connection
CONB$$	Build a short connect block for CON$ macro
CONL$$	Build a long connection block for CON$ macro
DSC$	Disconnect a logical link
GLN$	Get local node information
GND$	Get data from network data queue
OPN$	Access the network
REC$	Receive data over a logical link
REJ$	Reject logical link connect request
SND$	Send data over a logical link
SPA$	Specify a user AST routine
XMI$	Send interrupt message over a logical link

2.5.1 Common Argument Definitions

This section defines commonly-used arguments for intertask communication macros.

- *label*

  has the following meanings, depending on the macro type:

  BUILD type: *label* is a symbolic name associated with the location of the argument block.

EXECUTE type: *label* can represent one of two values:

The *label* of the BUILD macro that supplies arguments for the current EXECUTE macro. You can override any arguments that the BUILD macro defines by reentering them after *label* in the EXECUTE macro.

The *label* of an area of memory that will contain the arguments that you specify in the current EXECUTE macro.

- *status*

  unless noted otherwise, is the address of an optional 2-word status block that contains completion status information on return from the macro. If specified, this block will contain the following values when the macro completes:

  Word 0: Byte 0 = Error/completion code

  Byte 1 = 0

  Word 1: 0

  Each macro description lists the error/completion codes for that macro.

- *out,outlen*

  define optional user data to send with certain macros. These are optional arguments, but are always paired; use both or omit both.

  out is the octal starting address of a buffer that contains optional user data you can send on some operations.

  outlen is the length in decimal bytes of the 1- to 16.-byte message to send.

ABT$
(Abort a Logical Link)

2.5.2 ABT$ — Abort a Logical Link

Use:

Issue ABT$ from either task to abort a logical link. ABT$ immediately aborts all pending transmits and receives, disconnects the link, and frees the LUN assigned to the logical link. When you issue ABT$, you can send 1 to 16. bytes of user data to the task from which you are disconnecting (see the *out,outlen* arguments).

Formats:

label: ABT[W]$ *lun*,[*efn*],[*status*],[*ast*][,<*out,outlen*>]

ABT[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*out,outlen*>]

ABT[W]$S *lun*,[*efn*],[*status*],[*ast*][,<*out,outlen*>]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical link to abort. If you initiated the connection, enter the LUN you used in the CON$ macro. If you accepted the connection, enter the LUN you used in the ACC$ macro.

* *efn*

specifies an optional event flag number to set when ABT$ completes.

* *status*

specifies completion status information on return from ABT$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine to execute after ABT$ completes.

out,outlen

define optional user data to send. See the definition in Section 2.5.1.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The specified logical link has already been aborted or disconnected.
IE.BAD	The optional user data exceeds 16. bytes.
IE.IFC	LUN not assigned to NS:.
IE.NLN	No logical link has been established on the specified LUN.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.SPC	Invalid buffer argument; the optional user data buffer (*out*) is outside the user task address space.

ACC$
(Accept Logical Link Connect Request)

2.5.3 ACC$ — Accept Logical Link Connect Request

Use:

Issue ACC$ from the target task to establish a logical link with the source task. When you issue ACC$, you can send 1 to 16. bytes of user data to the source task (see the *out,outlen* arguments).

Formats:

label: ACC[W]$ *lun*,[*efn*],[*status*],[*ast*],<*mail*,[*mailen*],
[*out*,*outlen*]>[,NOFLOW]

ACC[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*],<[*mail*],[*mailen*],
[*out*,*outlen*]>[,NOFLOW]

ACC[W]$S *lun*,[*efn*],[*status*],[*ast*],<*mail*,[*mailen*],
[*out*,*outlen*]>[,NOFLOW]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

assigns the logical link number. Use this LUN to refer to this logical link in any subsequent REC$, SND$, XMI$, ABT$, or DSC$ macro.

* *efn*

specifies an optional event flag number to set when ACC$ completes.

* *status*

specifies completion status information on return from ACC$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine execute after ACC$ completes.

mail

is the address of the connect block sent by the source task and retrieved by GND$. Specify the same address for this and the GND$ *mail* argument. The connect block information is required to establish the connection.

mailen

is the length of the connect block in decimal bytes. The default value is 98. bytes (N.CBL), the short connect block length, not including optional data. For a long connect block, specify 178. bytes (M.CBL), the long connect block length, not including optional data.

out,outlen

define optional user data to send. See the definition in Section 2.5.1.

Flag:

NOFLOW

disables flow control for incoming messages addressed to the task that issued ACC$. Omitting NOFLOW establishes flow control for incoming messages. You can enable or disable flow control independently at either end of the link. Use the NOFLOW option with caution (see Section 1.2.12).

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The task that requested the connection has aborted or requested a disconnect before the connection could complete.
IE.ALN	A logical link has already been established on the specified LUN.
IE.BAD	Either the temporary link address in the connect block sent by the source task is invalid, or the optional user data buffer length (*outlen*) exceeds 16. bytes.
IE.IFC	LUN not assigned to NS:.

IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.RSU	System resources needed for the logical link are not available.
IE.SPC	Invalid buffer argument; either the pending connect block (*mail*) or the optional user data buffer (*out*) is not word aligned, or one of them is outside the user task address space.

CLS$
(End Task Network Operations)

2.5.4 CLS$ — End Task Network Operations

Use:

Issue CLS$ from either task to end that task's network activity, abort its logical links, and free its network LUNs. If the CLS$ call occurs when data remains in the task's network data queue, network software:

- Reschedules the task if pending connect requests arrived while the task was active. The task receives these connect requests when it restarts. There is a limit of one retry and a timeout period of approximately 15 seconds.
- Rejects connect requests that arrived while the task was inactive.
- Discards interrupt, user disconnect, user abort, or network abort messages.

Formats:

label: CLS[W]$ [*lun*],[*efn*],[*status*][,*ast*]

CLS[W]$E *label*,[*lun*],[*efn*],[*status*][,*ast*]

CLS[W]$S [*lun*],[*efn*],[*status*][,*ast*]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical unit number of the network data queue. Use the same LUN you assigned in the OPN$ macro.

* *efn*

specifies an optional event flag number to set when CLS$ completes.

* *status*

specifies completion status information on return from CLS$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine to execute after CLS$ completes.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.IFC	LUN not assigned to NS:.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.

CON$
(Request Logical Link Connection)

2.5.5 CON$ — Request Logical Link Connection

Use:

Issue CON$ from the source task to request a logical link with the target task. Before issuing CON$, you must build a connect block (see Section 2.5.6 or 2.5.7) that CON$ can pass to the target node.

When a remote system receives a connect request, it checks the remote task. If the task is currently installed and inactive, the system automatically loads and activates it. The target task must issue a GND$ macro call to retrieve the connect block information. The task evaluates the connect request and either accepts or rejects it.

You can send 1 to 16. bytes of user data to the target task and/or receive 1 to 16. bytes of user data from the target task when it accepts/rejects your connect request.

Formats:

label: CON[W]$ *lun*,[*efn*],[*status*],[*ast*],<*conbl*,[*conblen*],
[*out*,*outlen*],[*in*,*inlen*]>[,NOFLOW]

CON[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*],<*conbl*,[*conblen*],
[*out*,*outlen*],[*in*,*inlen*]>[,NOFLOW]

CON[W]$S *lun*,[*efn*],[*status*],[*ast*],<*conbl*,[*conblen*],
[*out*,*outlen*],[*in*,*inlen*]>[,NOFLOW]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

assigns the logical link number. Use this LUN to refer to this logical link in any subsequent REC$, SND$, XMI$, ABT$, or DSC$ macro.

* *efn*

specifies an optional event flag number to set when CON$ completes.

* *status*

is the address of an optional 2-word status block that contains completion status information on return from CON$. If specified, this block will contain the following values when CON$ completes:

Word 0: Byte 0 = Error/completion code (see the list that follows)

Byte 1 = 0

Word 1: Byte 0 = Contents depend on error completion code in word 0, byte 0 (see the list that follows)

Byte 1 = 0

This list shows the error/completion codes that you can receive in word 0, byte 0 and the corresponding contents of word 1, byte 0:

Error/Completion Code Word 0, Byte 0	**Word 1, Byte 0**
IS.SUC Connection accepted	Received byte count (0 if no data is received)
IS.DAO Connection accepted with data overrun	Received byte count (0 if no data is received)
IE.DAO Connection rejected by user with data overrun	Received byte count (0 if no data is received)
IE.URJ Connection rejected by user	Received byte count (0 if no data is received)
IE.NRJ Connection rejected by DECnet	Reason for rejection (refer to Appendix A)
All other cases	0

ast

is the address of an optional user-written AST routine to execute after CON$ completes.

conbl

is the address of the connect block built using CONL$$ or CONB$$. This block must start on an even byte (word) boundary.

conblen

is the length of the connect block in decimal bytes. If you omit this value, the CON$ macro uses the short connect block length, 72. bytes (N.RQL). To use a long connect block, specify the long connect block length, 152. (M.RQL).

out,outlen

define optional user data to send. See the definition in Section 2.5.1.

in,inlen

define the buffer to receive optional user data from the target task. These are paired optional arguments; use both or omit both. If you omit these arguments and the target task sends user data, a data overrun status code (IS.DAO or IE.DAO) will be returned.

* *in* is the octal address of the buffer.

inlen is the buffer length in decimal bytes (1 to 16.).

Flag:

NOFLOW

disables flow control for this end of the link. Omitting NOFLOW establishes flow control at this end of the link. You can choose to enable or disable flow control independently at each end of the link. Use the NOFLOW option with caution (see Section 1.2.12).

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IS.DAO	The macro completed successfully; the target task accepted the connection. However, the target task sent back some optional user data when it accepted the connect request, which was lost.
IE.ALN	A logical link has already been established on the specified LUN.
IE.BAD	Either the optional user data buffer exceeds 16. bytes, or the field length count in the connect block is too large.
IE.DAO	The connection was rejected and some optional user data sent from the target task when it rejected your connect request was lost.
IE.IFC	LUN not assigned to NS:.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.NRJ	The network rejected the connection (see the reject reason codes in Appendix A).
IE.PRI	The local node is shutting down. No logical link can be established.
IE.RSU	System resources needed for the logical link are not available.
IE.SPC	Invalid buffer argument; either the connect block (*conbl*) is not word aligned, or the optional user data buffers (*in* or *out*) are outside the user task address space.
IE.URJ	The remote user task rejected the connection.

CONB$$
(Build Connect Block (Short))

2.5.6 CONB$$ — Build Connect Block (Short)

Use:

Issue CONB$$ from the source task to build a 72.-byte connect block. The CON$ macro call passes this outgoing connect block to the target task. The connect block contains the target node name, destination descriptor, and, optionally, explicit access control information. The target task can use this information to determine whether to accept (ACC$) or reject (REJ$) the connect request.

To include explicit access control information, include the *rqid*, *pass*, and, optionally, *accno* arguments in the macro. You can omit the explicit access control information if you already included it in an alias node name or if you use proxy access. The target system verifies access control information according to its system conventions. If the target node uses and has enabled access verification, it performs verification before passing the connect request to the target task.

For more information on access control verification, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on aliases, refer to the *DECnet–RSX Guide to User Utilities* or the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on proxy access, refer to the *DECnet–RSX Guide to Network Management Utilities*.

Format:

CONB$$ [*node*],[*obj*],[*fmt*,<*descrip*>],[*rqid*],[<*pass*>][,*accno*]

Arguments:

node

is the name of the target node. The name must have 1 to 6 alphanumeric characters, including at least 1 alphabetic character.

The *obj*, *fmt*, and *descrip* arguments comprise the destination descriptor. You must specify this information in order to access the task.

obj

is the target task's object type. The object type for a named object is 0. The object type for a numbered object is in the range 1 to 127. for a DECnet task or 128. to 255. for a user task. Refer to Appendix B for a list of object type codes.

Privileged users can define their own object types; for information, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.

fmt

is the descriptor format type. To connect to a named object, specify 0 for the descriptor format type and specify the *descrip* argument. To connect to a numbered object, specify 1 and omit the *descrip* argument.

descrip

is the target task name (1 to 16. ASCII characters). Specify this argument only if you specified 0 for the *fmt* argument.

The *rqid*, *pass*, and *accno* arguments comprise explicit access control information that specifies an account on the remote node.

rqid

is the user ID (1 to 16. ASCII characters).

pass

is a 1 to 8.-byte password. To enter an ASCII (as opposed to binary) password, precede each character of the password with an apostrophe (') and separate the characters with commas. For example, enter the password PAS as 'P,'A,'S.

accno

is your account number at the remote node or process (1 to 16. ASCII characters).

NOTE

During task execution, you can dynamically supply or modify values for the connect block fields, using the offsets in Table 2–2. You must use this method to supply non-ASCII data for a field that normally requires ASCII data.

You can also choose not to issue CONB$$ and instead allocate a 72. byte block of storage. Issue the CRBDF$ call to define the offsets, with which you fill in the connect block.

Remember that a successful connect request requires that the connect block contain certain fields, whether you enter the values as arguments to the CONB$$ macro or dynamically during task execution.

Connect Block:

Table 2–2 describes the connect block's symbolic offsets. Figure 2–1 is an example of a 72.-byte connect block.

Table 2–2: CONB$$ Connect Block Symbolic Offsets

Symbolic Offset	Length in Bytes	Contents
		DESTINATION DESCRIPTOR
N.RND*	6.	Remote node name with trailing blanks
N.RFM	1.	Destination descriptor format type: 0 or 1
N.ROT	1.	Destination object type: 0–255.
		Descriptor Field for Format 0
		18. Not used
		Descriptor Fields for Format 1
		N.RDEC* 2. Destination task name length (equal to or less than 16. bytes)
		N.RDE* 16. Destination task name
		ACCESS CONTROL INFORMATION
N.RIDC*	2.	User ID length (equal to or less than 16. bytes)
N.RID*	16.	User ID
N.RPSC*	2.	Password length (equal to or less than 8. bytes)
N.RPS*	8.	Password
N.RACC*	2.	Account number length (equal to or less than 16. bytes)
N.RAC*	16.	Account number
N.RQL = 72.		

* These symbolic offsets are guaranteed to be even (word aligned).

Figure 2–1 illustrates the connect block that the following call builds:

```
CONB$$ TACOMA,0,1,<RECVR>,BLOGGS,<'P,'A,'S>
```

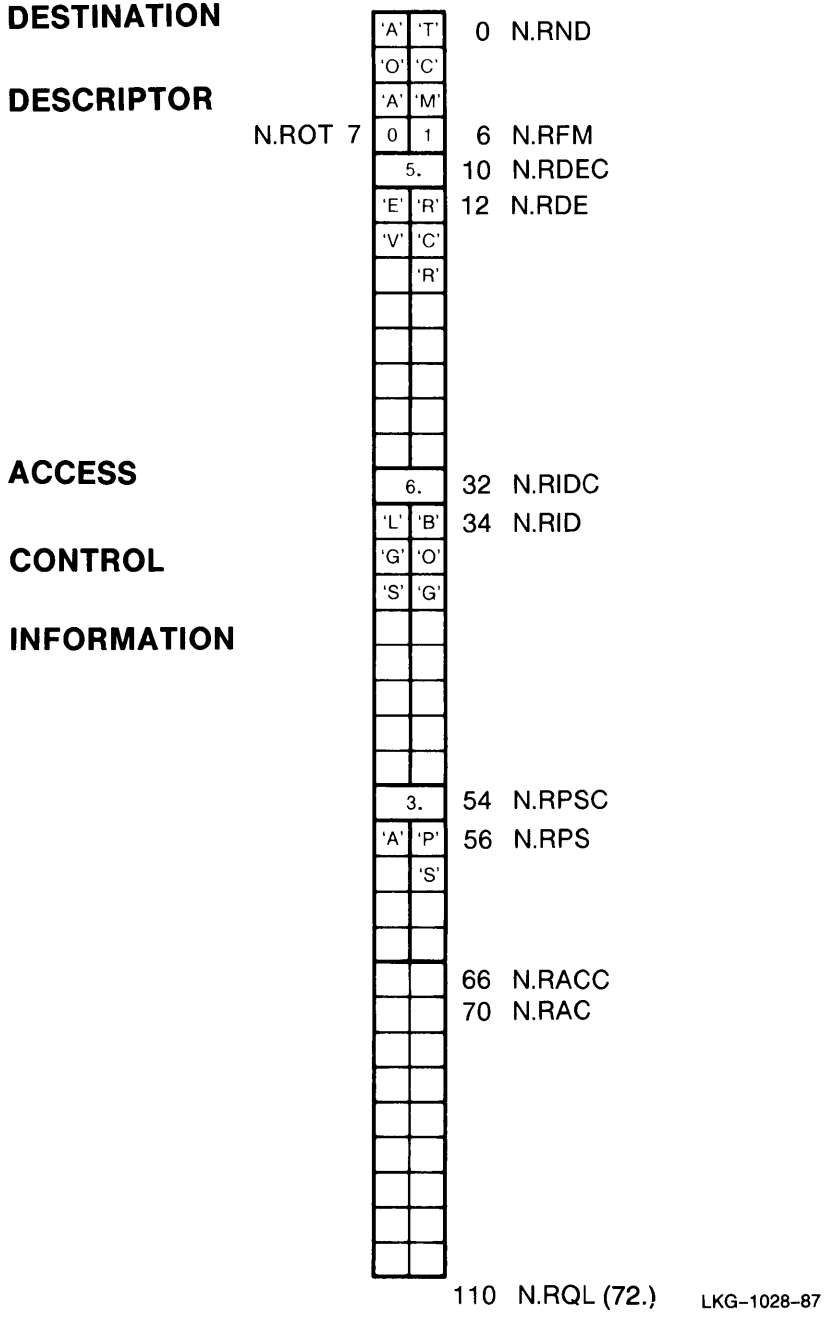
The connect block contains the following values:

Field	Value
Destination node	TACOMA, an RSX node
Object type	0 (named object)
Descriptor format type	1
Destination task name length	5
Destination task name	RECVR
User ID length	6
User ID	BLOGGS
Password length	3
Password	PAS

The account number length and account number are omitted because RSX target systems do not require them.

The call supplies explicit access control information. It could omit that information when an alias node name contains the user ID and password, or to use proxy access. The access control fields in the figure would then be empty. The offsets are in octal notation.

Figure 2–1: Outgoing CONB$$ Connect Block



LKG-1028-87

CONL$$
(Build Connect Block (Long))

2.5.7 CONL$$ — Build Connect Block (Long)

Use:

Issue CONL$$ from the source task to build a 152.-byte connect block. The CON$ macro call passes this outgoing connect block to the target task. The connect block contains the node name, destination descriptor, and, optionally, explicit access control information. The target task can use this information to determine whether to accept (ACC$) or reject (REJ$) the connect request.

To specify explicit access control information, you call the following associated macros:

- CNID$$ lets you specify a user ID
- CNPS$$ lets you specify a password
- CNAC$$ lets you specify an account number

To omit one or more of the access control fields, simply omit calling the macro. You need not issue a separate .MCALL directive for each; the .MCALL directive for CONL$$ calls CONL$$ and the three associated macros. You can omit the access control information if you already included it in an alias node name or if you use proxy access. The target system verifies access control information according to its system conventions. If the target node uses and has enabled access verification, it performs verification before passing the connect request to the target task.

For more information on access control verification, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on aliases, refer to the *DECnet–RSX Guide to User Utilities* or the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on proxy access, refer to the *DECnet–RSX Guide to Network Management Utilities*.

Format:

CONL$$ [*node*],[*obj*],[*fmt*,<*descrip*>]

CNID$$ [*rqid*]

CNPS$$ [<*pass*>]

CNAC$$ [*accno*]

Arguments for CONL$$:

node

is the name of the target node. The name must have 1 to 6 alphanumeric characters, including at least 1 alphabetic character.

The *obj*, *fmt*, and *descrip* arguments comprise the destination descriptor. You must specify this information in order to access the target task.

obj

is the target task's object type. The object type for a named object is 0. The object type for a numbered object is in the range of 1 to 127. for a DECnet task or 128. to 255. for a user task. Refer to Appendix B for a list of object type codes.

Privileged users can define their own object types; for information, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.

fmt

is the descriptor format type. To connect to a named object, specify 0 for the descriptor format type and specify the *descrip* argument. To connect to a numbered object, specify 1 and omit the *descrip* argument.

descrip

is the target task name (1 to 16. ASCII characters). Specify this argument only if you specified 0 for the *fmt* argument.

CONL$$

Argument for CNID$$:

rqid

is the user ID (1 to 39. ASCII characters).

Argument for CNPS$$:

pass

is a 1 to 39.-byte password. To enter an ASCII (as opposed to binary) password, precede each character of the password with an apostrophe (') and separate the characters with commas. For example, enter the password RADIO as 'R,'A,'D,'I,'O.

Argument for CNAC$$:

accno

is your account number at the remote node or process (1 to 39. ASCII characters).

NOTE

You can choose to create the entire connect block or specify any of the CONL$$, CNID$$, CNPS$$, or CNAC$$ arguments dynamically during task execution. In addition, to specify non-ASCII data for an argument that normally requires ASCII data, you must do so dynamically. You can also modify any connect block field this way.

To create the connect block dynamically, reserve a 152.-byte block of storage, which equals the M.RQL length (see Table 2–3); to specify any connect block field dynamically, leave the argument blank in the macro call. Issue the CRBDF$ call to define the connect block symbolic offsets listed in Table 2–3. During task execution, use these offsets to specify or modify the connect block information.

A successful connect request (CON$) requires all of the necessary connect block fields, whether you put them in the macro arguments or enter them dynamically during task execution.

Connect Block:

Table 2–3 lists the connect block symbolic offsets. Figure 2–2 illustrates a sample connect block.

Table 2–3: CONL$$ Connect Block Symbolic Offsets

Symbolic Offset	Length in Bytes	Contents
		DESTINATION DESCRIPTOR
M.RND*	6.	Remote node name with trailing blanks
M.RFM	1.	Destination descriptor format type: 0 or 1
M.ROT	1.	Destination object type: 0–255.
		Descriptor Field for Format 0
		18. Not used
		Descriptor Fields for Format 1
		M.RDEC* 2. Destination task name length (equal to or less than 16. bytes)
		M.RDE* 16. Destination task name
		EXPLICIT ACCESS CONTROL INFORMATION
M.RIDC*	2.	User ID length (equal to or less than 39. bytes)
M.RID*	39.	User ID
	1.	Not used
M.RPSC*	2.	Password length (equal to or less than 39. bytes)
M.RPS*	39.	Password
	1.	Not used
M.RACC*	2.	Account number length (equal to or less than 39. bytes)
M.RAC*	39.	Account number
	1.	Not used
M.RQL = 152.		

* These symbolic offsets are guaranteed to be even (word aligned).

Figure 2–2 illustrates the connect block that the following call builds:

```
CONL$$ GROTON,0,1,<RECEIVER>
CNID$$ EDGAR
CNPS$$ <'R,'A,'D,'I,'O,'S,'T,'A,'T,'I,'O,'N>
CNAC$$
```

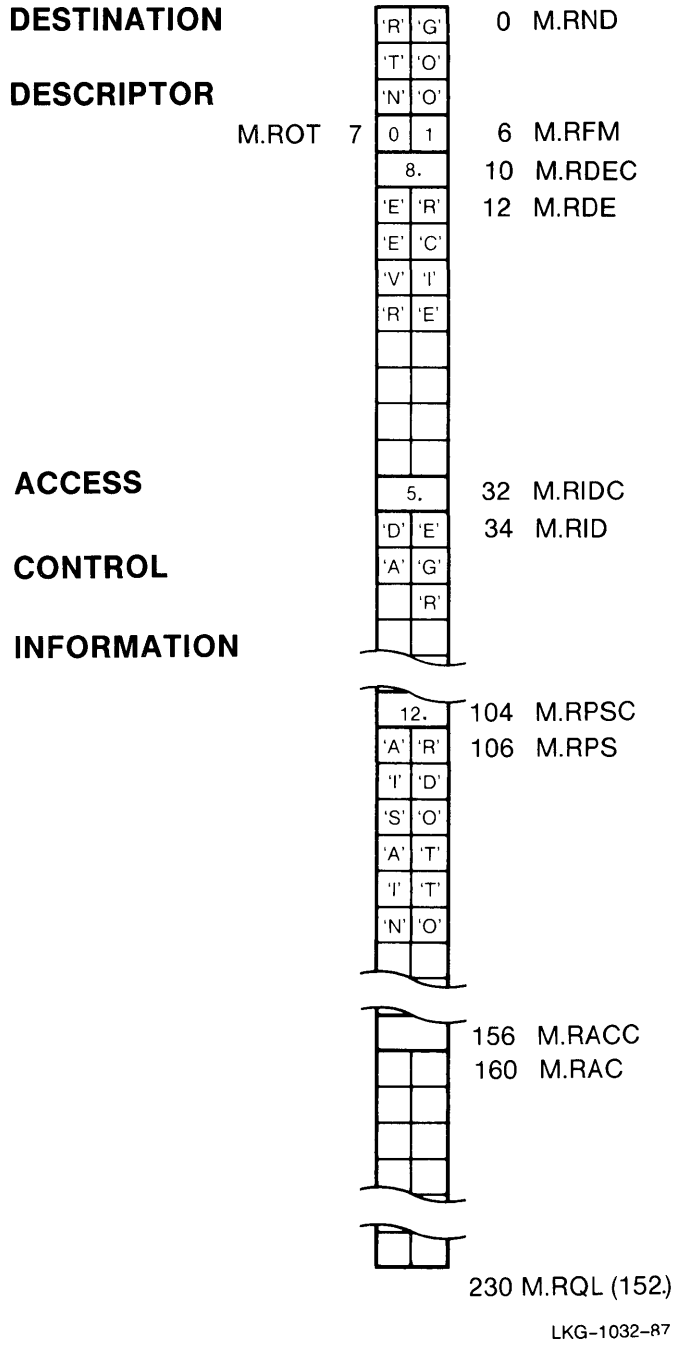
The connect block contains the following values:

Field	Value
Destination node	GROTON, an RSX node
Object type	0 (named object)
Descriptor format type	1
Destination task name length	8
Task name	RECEIVER
User ID length	5
User ID	EDGAR
Password length	12
Password	RADIOSTATION

The account number length and account number are omitted because RSX target systems do not require account numbers.

The call supplies explicit access control information. If, instead, you have defined the access control information in an alias node name, or if you use proxy, the access control information fields in the figure would be empty. The offsets are in octal notation.

Figure 2–2: Outgoing CONL$$ Connect Block



DSC$
(Disconnect a Logical Link)

2.5.8 DSC$ — Disconnect a Logical Link

Use:

Issue DSC$ from either task to disconnect the logical link and free the logical unit number. Unlike ABT$ (Section 2.5.2), DSC$ causes all pending transmits to complete before disconnecting the link. While these transmits are completing, the task continues to receive messages. When the last transmit has completed, each pending receive is aborted with an IE.ABO status code in the I/O status block. With DSC$, you can send 1 to 16. bytes of user data to the task from which you are disconnecting (see the *out,outlen* arguments).

Formats:

label: DSC[W]$ *lun*,[*efn*],[*status*],[*ast*][,<*out,outlen*>]

DSC[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*out,outlen*>]

DSC[W]$S *lun*,[*efn*],[*status*],[*ast*][,<*out,outlen*>]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical link to disconnect. If you initiated the connection, enter the LUN you used in the CON$ macro. If you accepted the connection, enter the LUN you used in the ACC$ macro.

* *efn*

specifies an optional event flag number to set when DSC$ completes.

* *status*

specifies completion status information on return from DSC$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine to execute after DSC$ completes.

out,outlen

define optional user data to send. See the definition in Section 2.5.1.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The specified logical link has already been aborted or disconnected.
IE.BAD	The optional user data exceeds 16. bytes.
IE.IFC	LUN not assigned to NS:.
IE.NLN	No logical link has been established on the specified LUN.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.
IE.SPC	Invalid buffer argument; the optional user data buffer (*out*) is outside the user task address space.

GLN$
(Get Local Node Information)

2.5.9 GLN$ — Get Local Node Information

Use:

Issue GLN$ from either task to place the name and default NSP segment size of the local node in a specified buffer.

Getting the local node name can be helpful if two tasks on the same node use the network interface to communicate. Each task can issue GLN$ and use the returned local node name as the destination in a connect request. A task that displays the local node name can also use GLN$.

The default NSP segment size tells you how NSP segments data transmitted on a logical link. By knowing the default NSP segment size, you can adjust the length of message blocks to transmit for most efficient use of transmit buffers (large data buffers).

Formats:

label: GLN[W]$[*lun*],[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

GLN[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

GLN[W]$S [*lun*],[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

* *efn*

specifies an optional event flag number to set when GLN$ completes.

* *status*

is the address of an optional 2-word status block that contains completion status information on return from GLN$. If *status* is specified, the contents of word 1 depend on the error/completion code returned in word 0, byte 0 (word 0, byte 1 is always 0).

Contents in Word 0, Byte 0	Contents of Word 1
IS.SUC (1) or IE.DAO (–13)	Number of bytes transferred to the user buffer
IE.*xxx* (excluding IE.DAO, *xxx* refers to IE.NNT, IE.PRI, IE.SPC)	0

ast

is the address of an optional user-written AST routine to execute after GLN$ completes.

* *buf*

is the address of the buffer to contain the received data. This buffer must start on an even byte (word) boundary.

buflen

is the length of the buffer to contain the received data. The buffer length determines the data returned, as follows.

Length	Returned Data
6 bytes	Local node name, left justified and in ASCII. Names with fewer than 6 bytes are padded with spaces.
8 bytes	Local node name, default NSP segment size.
10 bytes	Local node name, default NSP segment size, node number.

The first six bytes contain the local node name. The next two bytes contain the default segment size. The last two bytes contain the local node number in the lower 10 bits and the local area number in the higher 6 bits.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.DAO	Data overrun. The network data was longer than the specified buffer. As much data as fits into the buffer is transferred to it; any remaining data is lost.
IE.IFC	LUN not assigned to NS:.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.
IE.SPC	Invalid buffer argument; the buffer specified to receive network data (*buf*) is outside the user task address space.

GND$
(Get Network Data)

2.5.10 GND$ — Get Network Data

Use:

Issue GND$ from either task to get data from that task's network data queue and store it in a mail buffer. You specify the buffer in the *mail* and *mlen* parameters. The status block identifies what type of message the call retrieved. The status block identifies one of the following unsolicited message types in word 0, byte 1:

Connect request	NT.CON
Interrupt message	NT.INT
User disconnect notice	NT.DSC
User abort notice	NT.ABT
Network abort notice	NT.ABO

You can use the SPA$ macro (Section 2.5.15) to get a count of data items in the network data queue. If the queue is empty, GND$ completes with an error (IE.NDA), even if you use the GND[W] form.

If GND$ retrieves a connect request, it writes the accompanying connect block information to the *mail* buffer. You can use a long or short connect block, depending on the length of the user IDs, passwords, and accounts you expect to receive. For information about the incoming connect block, see the "Connect Block" section of this call description.

Formats:

label: GND[W]$ [*lun*],[*efn*],[*status*],[*ast*],

$$\left\{\begin{array}{l} <mail,mlen> \\ <mail,mlen,mask>,\text{NT.TYP} \\ ,\text{NT.LON} \\ <,,mask>,\text{NT.LON} \end{array}\right\}$$

GND$

GND[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*],

{ <*mail*,*mlen*> | <*mail*,*mlen*,*mask*> ,NT.TYP | ,NT.LON | <,,*mask*> ,NT.LON }

GND[W]$S [*lun*],[*efn*],[*status*],[*ast*],

{ <*mail*,*mlen*> | <*mail*,*mlen*,*mask*> ,NT.TYP | ,NT.LON | <,,*mask*> ,NT.LON }

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical unit number assigned to the network data queue. Use the LUN that you specified in OPN$.

* *efn*

specifies an optional event flag number to set when GND$ completes.

* *status*

is the address of an optional 2-word status block that contains completion status information on return from GND$. Refer to Table 2–4 for a summary of the status block contents after GND$.

ast

is the address of an optional user-written AST routine to execute after GND$ completes.

mail,mlen

define the task mail buffer to receive the network data or connect block on return from GND$. You must specify these arguments unless you use the NT.TYP and NT.LON flags. Refer to Table 2–5 for a list of the short connect block contents and to Table 2–6 for a list of the long connect block contents.

* *mail*

is the octal address of the buffer, which must start on an even byte (word) boundary.

mlen

is the length of the buffer in decimal bytes. The incoming data is written to the buffer according to the offsets of the connect block type that you specified in the OPN$ call.

You can allocate a mail buffer that is equal to, smaller than, or larger than the expected connect block and optional data. To receive an entire connect block, allocate space according to the connect block type that you specified in the OPN$ macro call:

Short connect block	98. bytes (N.CBL)
Long connect block	178. bytes (M.CBL)

You can add space for optional data:

Optional data	Up to 16. bytes
Optional data length field	2. bytes

Network software writes the retrieved information to the buffer field by field, according to the offsets of the specified connect block type. If the mail buffer and the incoming connect block are different sizes, the following results occur.

Mail Buffer Size	Result
You allocate a buffer that is larger than the incoming connect block.	No error occurs.
You allocate a buffer that is smaller than a full connect block.	Connect block data is written field by field into the buffer until no more fits. A data overrun (IS.DAO) completion status results, even if all the received data fits into the buffer.
You allocate a buffer for receiving a short connect block and instead receive a long connect block.	If the incoming data fits according to the short connect block offsets, you get all the data, but a data overrun (IS.DAO) completion status results. If the data in any incoming field exceeds the size of the analogous receiving field, the data in that field is lost. The length value for the field becomes 0, and a data overrun (IS.DAO) completion status results.

mask

specifies the data type to select from the network data queue. Normally, GND$ returns items from the network data queue on a first-in, first-out basis. However, *mask* lets you select the first item on the queue that matches a specific message type and/or LUN. Enter one of the following combinations for the *mask* argument.

Message Type (Byte 0)	Logical Unit Number (Byte 1)
NT.CON (connect request)	0 (Selects the first LUN of message type NT.CON)
NT.INT (interrupt)	0 or LUN
NT.DSC (user disconnect)	0 or LUN
NT.ABT (user abort)	0 or LUN
NT.ABO (network abort)	0 or LUN
0 (Selects any message type on the specified LUN).	LUN

For example, to select the first disconnect message (NT.DSC) on LUN 3 from the network data queue, code the *mask* argument as 3*256. + NT.DSC.

Specifying 0 in byte 1 returns the first message of the type specified in byte 0, regardless of LUN.

Flags:

NT.TYP

indicates a *mask* argument requesting a specific message type and/or LUN. Always use NT.TYP when specifying *mask* with *mail* and *mlen*.

If you use NT.TYP in a BUILD type GND$, you must also use it in any subsequent EXECUTE type GND$.

NT.LON

supports dynamic assignment of mail buffer space. Specifying NT.LON with GND$ returns information about the first message in the network data queue without removing the message from the queue or placing it in the mail buffer. With NT.LON, the status block returns the message type in word 0, byte 1 and the message length in word 1, byte 0. You cannot use *mail*, *mlen*, and NT.TYP with NT.LON.

If you use NT.LON in a BUILD type GND$, you must also use it in any subsequent EXECUTE type GND$.

Table 2–4: Status Block Contents After GND$

If GND$ completes successfully with NT.LON omitted:

Status Word 0		Status Word 1	
Byte 0	**Byte 1**	**Byte 0**	**Byte 1**
IS.SUC or IS.DAO or IE.DAO	NT.CON Connect request	Number of bytes in connect block.	Access verification (1) and privileged code: VS.NPV = Requesting user is nonprivileged. VS.PRV = Requesting user is privileged. VZ.NVD = Verification was not done. (2) VE.FAI = Verification failed. (3)
IS.SUC or IE.DAO	NT.INT Interrupt message	Number of bytes (1–16) in optional message. If 0, no message was received.	LUN over which the interrupt message was received.
	NT.DSC User disconnect	Number of bytes (1–16) in optional message. If 0, no message was received.	LUN over which the user disconnect message was received.
	NT.ABT User abort	Number of bytes (1–16) in optional message. If 0, no message was received.	LUN over which the network abort message was received.
IS.SUC or IE.DAO	NT.ABO Network abort	Reason for network abort (See codes in Appendix A).	LUN over which the notice was received.

If GND$ completes successfully with NT.LON specified:

Status Word 0		Status Word 1	
Byte 0	**Byte 1**	**Byte 0**	**Byte 1**
IS.SUC or IE.DAO	NT.*xxx* (type of first item in queue)	Number of bytes in first item in network data queue.	0

Table 2–4 (Cont.): Status Block Contents After GND$

If GND$ completes with an error other than IE.DAO (–13):

Status Word 0		Status Word 1	
Byte 0	**Byte 1**	**Byte 0**	**Byte 1**
IE. *xxx*	0	0	0

1. If access verification is enabled, the Network Verification Program at the target node evaluates access control information in the connect request before passing the request to the target task's network data queue.

2. The verification task was not installed on the target node, it was set to OFF with the NCP SET EXECUTOR VERIFICATION command, or the proper access control file was not available.

3. The account is not in the system account file, the password does not match the one in the file, or the object is set to inspect.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IS.DAO	The macro completed successfully, but some returned optional data was lost.
IE.DAO	Data overrun. The network data was longer than the mail buffer. As much data as will fit into the mail buffer is transferred to it; any remaining data is lost.
IE.IFC	LUN not assigned to NS:.
IE.NDA	There is no data in the network data queue to return.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.
IE.SPC	Invalid buffer argument; the buffer assigned to receive network data (*mail*) is not word aligned or is outside the user task address space.

Connect Block:

This section includes Table 2–5, which lists the contents of the short connect block, and Table 2–6, which lists the contents of the long connect block. It also includes Figure 2–3, an example of an incoming connect short block.

The source descriptor differs according to the source system type. If the source is an RSX system

- and the connect request does not include proxy information, you receive a Format 1 source desriptor containing the ASCII source task name.
- and the connect request includes proxy information, you receive a Format 2 source descriptor containing the proxy information.

The access control information that GND$ returns differs according to whether access verification is set to ON or OFF for the task.

Table 2–5: Contents of Incoming Short Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents
N.CTL*	2.	Temporary logical link address (required by the network; do not modify)
N.SEGZ*	2.	NSP segment size (used by NSP to send message data to source)
		DESTINATION DESCRIPTOR (20.-byte total)
N.DFM	1.	Destination descriptor format type: 0,1
N.DOT	1.	Destination object type: 0–255.
		Descriptor Field for Format 0
		18. Not used

* These symbolic offsets are guaranteed to be even (word aligned).

Table 2–5 (Cont.): Contents of Incoming Short Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents		
		Descriptor Fields for Format 1		
		N.DDEC*	2.	Destination task name length (equal to or less than 16. bytes)
		N.DDE*	16.	Destination task name
		SOURCE DESCRIPTOR (26.-byte total)		
N.SND*	6.	Source node name (name of node requesting the connection; ASCII, with trailing blanks)		
N.SFM	1.	Source descriptor format type (format 0, 1, or 2)		
N.SOT	1.	Source object type (object type of task or process requesting the connection: 1–255. for format 0, or 0 for format 1)		
		Descriptor Field for Format 0		
			18.	Not used
		Descriptor Fields for Format 1		
		N.SDEC*	2.	Source descriptor length (equal to or less than 16. bytes)
		N.SDE*	16.	Source descriptor (ASCII)
		Descriptor Fields for Format 2		
		N.SGRP*	2.	Binary UIC group identifier
		N.SMEM*	2.	Binary UIC member identifier
		N.SDRC*	2.	Source descriptor length (equal to or less than 12. bytes)
		N.SDR*	12.	Source descriptor

* These symbolic offsets are guaranteed to be even (word aligned).

(continued on next page)

Table 2–5 (Cont.): Contents of Incoming Short Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents		
		ACCESS CONTROL INFORMATION (46.-byte total)		
		If no verification is performed		
		N.CIDC*	2.	User ID length (equal to or less than 16. bytes)
		N.CID*	16.	User ID
		N.CPSC*	2.	Password length (equal to or less than 8. bytes)
		N.CPS*	8.	Password
		N.CACC*	2.	Account number length (equal to or less than 16. bytes)
		N.CAC*	16.	Account number
		If verification is performed		
		N.CDEV	2.	Default device name
		N.CUNI	1.	Default device unit number
			1.	Not used
		N.CUIC	2.	Log-in UIC from account file
		N.CDDS	11.	Default directory string (0 if no default string)
			29.	Not used
N.CBL = 98. (not including optional data)				
		OPTIONAL DATA (18.-byte total)		
N.CDAC*	2.	Length of optional user data (equal to or less than 16. bytes; 0 if no optional data)		
N.CDA*	16.	Optional user data sent by source task (0 to 16. bytes)		

* These symbolic offsets are guaranteed to be even (word aligned).

Table 2–6: Contents of Incoming Long Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents
M.CTL*	2.	Temporary logical link address (required by the network; do not modify)
M.SEGZ*	2.	NSP segment size (used by NSP to send message data to source)
		DESTINATION DESCRIPTOR (20.-byte total)
M.DFM	1.	Destination descriptor format type: 0,1
M.DOT	1.	Destination object type: 0–255.
		Descriptor Field for Format 0
		18. Not used
		Descriptor Fields for Format 1
		M.DDEC* 2. Destination task name length (equal to or less than 16. bytes)
		M.DDE* 16. Destination task name
		SOURCE DESCRIPTOR (26.-byte total)
M.SND*	6.	Source node name (name of node requesting the connection; ASCII, with trailing blanks)
M.SFM	1.	Source descriptor format type (must be either format 0 or format 1)
M.SOT	1.	Source object type (object type of task or process requesting the connection: 1–255. for format 0, or 0 for format 1)
		Descriptor Field for Format 0
		18. Not used
		Descriptor Fields for Format 1
		M.SDEC* 2. Source descriptor length (equal to or less than 16. bytes)
		M.SDE* 16. Source descriptor (ASCII)

* These symbolic offsets are guaranteed to be even (word aligned).

(continued on next page)

Table 2–6 (Cont.): Contents of Incoming Long Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents		
		Descriptor Fields for Format 2		
		N.SGRP*	2.	Binary UIC group
		N.SMEM*	2.	Binary UIC member
		N.SDRC*	2.	Source descriptor length (equal to or less than 12. bytes)
		N.SDR*	12.	Source descriptor
		ACCESS CONTROL INFORMATION (126.-byte total)		
		If no verification is performed		
		M.CIDC*	2.	User ID length (equal to or less than 39. bytes plus 1 byte for an even byte count)
		M.CID*	39.	User ID
			1.	Not used
		M.CPSC*	2.	Password length (equal to or less than 39. bytes plus 1 byte for an even byte count)
		M.CPS*	39.	Password
			1.	Not used
		M.CACC*	2.	Account number length (equal to or less than 39. bytes plus 1 byte for an even byte count)
		M.CAC*	39.	Account number
			1.	Not used
		If verification is performed		
		M.CDEV	2.	Default device name
		M.CUNI	1.	Default device unit number
			1.	Not used

* These symbolic offsets are guaranteed to be even (word aligned).

Table 2–6: (Cont.) Contents of Incoming Long Connect Block

Symbolic Offset	Length in Decimal Bytes	Contents		
		M.CUIC	2.	Log-in UIC from account file
		M.CDDS	11.	Default directory string (0 if no default string)
			109.	Not used
M.CBL = 178. (not including optional data)				
		OPTIONAL DATA (18.-byte total)		
M.CDAC*	2.	Length of optional user data (equal to or less than 16. bytes; 0 if no optional data)		
M.CDA*	16.	Optional user data sent by source task (0 to 16. bytes)		

* These symbolic offsets are guaranteed to be even (word aligned).

Figure 2–3 is an example of an incoming connect block. The figure shows the connect block that the following macro created in the source task:

```
CONB$$ TACOMA,0,1,<RECVR>,BLOGGS,<'P,'A,'S>
```

The connect block contains the following values:

Field	Value
Destination descriptor format type	1
Destination object type	0 (named object)
Destination task name length	5
Destination task name	RECVR
Source node name	TACOMA
Source descriptor format type	1
Source object type	0 (named object)
Source descriptor length	6

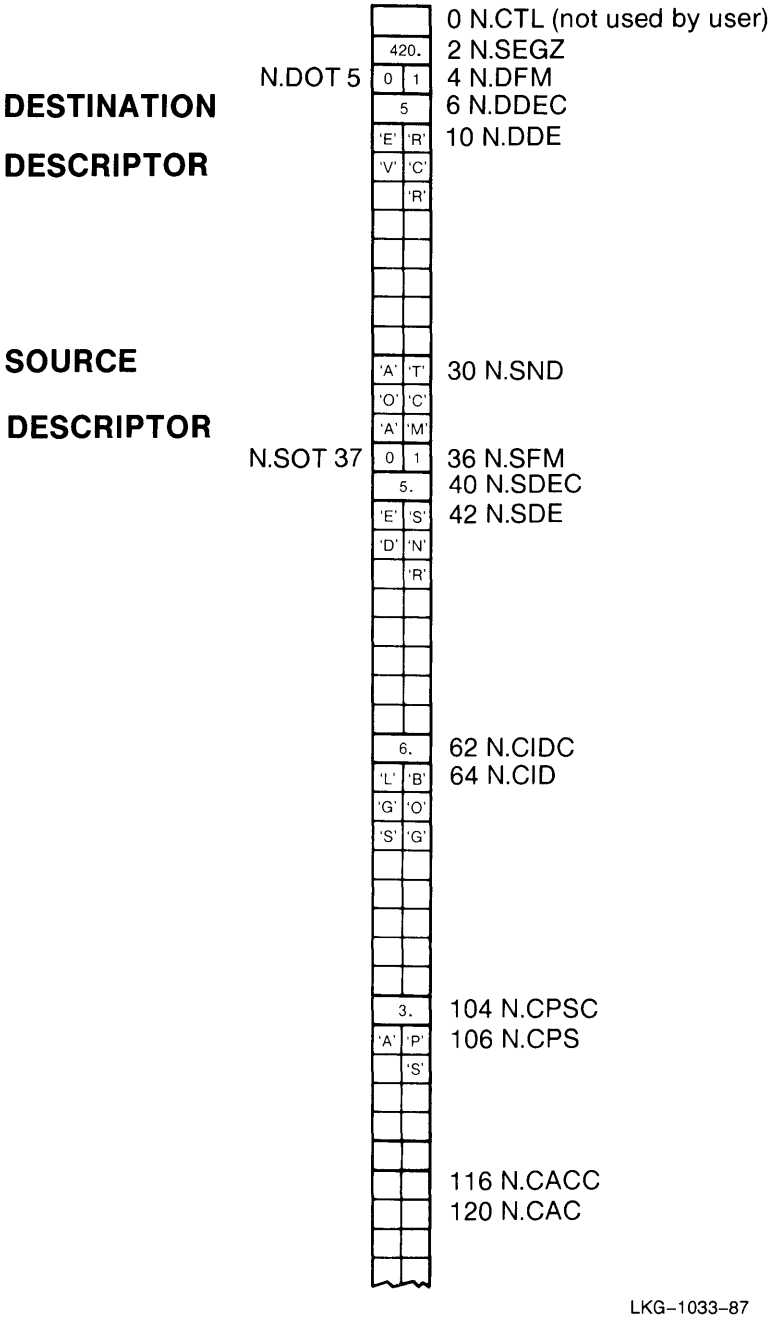
Field	Value
Source descriptor	SENDR
User ID length	6
User ID	BLOGGS
Password length	3
Password	PAS

The account number length and account number are omitted because RSX nodes do not require account numbers.

Because the call supplied explicit access control information (user ID and password), the incoming connect block contains that information. If outgoing proxy is enabled on the source node, the source descriptor contains the proxy information and the source descriptor type is format 2. If verification is on at the target node, the password is cleared out before the target task receives the connect block. All byte counts and values are in decimal notation.

This figure illustrates a short connect block. A long connect block has the same fields, but the access control information fields are longer, and the symbolic offset names are prefixed with M. instead of N.

Figure 2–3: Incoming Connect Block



LKG–1033–87

OPN$
(Access the Network)

2.5.11 OPN$ — Access the Network

Use:

Issue OPN$ to establish the task as an active network task and create the task's network data queue. Issue OPN$ before issuing any other intertask communication macro.

Formats:

label: OPN[W]$ [*lun*],[*efn*],[*status*],[*ast*][,<*links*[,*lrp*][,NT.LCB]>]

OPN[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*links*[,*lrp*][,NT.LCB]>]

OPN[W]$S [*lun*],[*efn*],[*status*],[*ast*][,<*links*[,*lrp*][,NT.LCB]>]

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

assigns a logical unit number to the task's network data queue. You can omit this argument if you have already assigned the LUN to NS: by defining the symbol .MBXLU in the user program or in a GBLDEF option at task build time (Section 1.2.1). Use this LUN in any subsequent GND$, SPA$, GLN$, REJ$, or CLS$ macro.

* *efn*

specifies an optional event flag number to set when OPN$ completes.

* *status*

specifies completion status information on return from OPN$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine to execute after OPN$ completes.

links

specifies the maximum number of simultaneous, active logical links within the task. When the number of active links equals the *links* value (255. maximum), the network rejects any incoming connect request. A value of 0 sets no limit as long as network resources are available. Zero is also the default.

To prevent access to your task, specify a *links* value of 1 and code the routine that processes the GND$ macro to reject all incoming connect requests. You can still use CON$ to establish outgoing links.

lrp

specifies the link recovery period. The link recovery period is the number of minutes that elapses from the time of a physical link failure until the network aborts the associated logical link. The *lrp* must be in the range of 0 through 32767(decimal).

When specifying an *lrp* value, remember that unless your task includes checkpoint capabilities, it is locked in memory until the link recovery period elapses if it has outstanding I/O when the link fails. This can cause serious delays for other system users who need to access the occupied area of memory.

Flag:

NT.LCB

specifies that the task transmits and receives long connect blocks that support 39.-character user IDs, passwords, and accounts. If a connect request for your task arrives with a short connect block, network software copies the information in each field to the corresponding field in the long format before passing it to your task. The NT.LCB value is 1.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.IFC	LUN not assigned to NS:.
IE.PRI	The network is being dismounted, or the user task has already accessed the network.
IE.RSU	System resources needed for the network data queue are not available.

REC$
(Receive Data over a Logical Link)

2.5.12 REC$ — Receive Data over a Logical Link

Use:

Issue REC$ from either task to receive message data over an established logical link and store it in a specified buffer.

Formats:

label: REC[W]$ *lun*,[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

REC[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*buf*,*buflen*>]

REC[W]$S *lun*,[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

specifies the logical link over which to receive data. If you initiated the connection, enter the LUN you used in the CON$ macro. If you accepted the connection, enter the LUN you used in the ACC$ macro.

* *efn*

specifies an optional event flag number to set when REC$ completes.

* *status*

specifies completion status information on return from REC$. See the definition in Section 2.5.1, but note this exception:

Word 1: Contains number of bytes received.

ast

is the address of an optional user-written AST routine to execute after REC$ completes.

* *buf*

is the address of the buffer to contain the received message data.

buflen

is the length of the receive buffer in bytes (8128. maximum).

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The logical link was disconnected during I/O operations.
IE.DAO	Data overrun. More message data was transmitted than requested. As much data as will fit into the receive buffer is transferred to it; any remaining data is lost.
IE.IFC	LUN not assigned to NS:.
IE.NLN	No logical link has been established on the specified LUN.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.SPC	Invalid buffer argument; either the data buffer (*buf*) is outside the user task address space, or the buffer length (*buflen*) exceeds 8128. bytes.

REJ$
(Reject Logical Link Connect Request)

2.5.13 REJ$ — Reject Logical Link Connect Request

Use:

Issue REJ$ from the target task to reject a logical link connect request. When you issue REJ$, you can send 1 to 16. bytes of user data to the requesting task (see the *out,outlen* arguments).

Formats:

label: REJ[W]$ [*lun*],[*efn*],[*status*],[*ast*],<*mail*,[*mailen*][,*out*,*outlen*]>

REJ[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*],<*mail*,[*mailen*][,*out*,*outlen*]>

REJ[W]$S [*lun*],[*efn*],[*status*],[*ast*],<*mail*,[*mailen*][,*out*,*outlen*]>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical unit number of the network data queue. Use the same LUN you assigned in the OPN$ macro.

* *efn*

specifies an optional event flag number to set when REJ$ completes.

* *status*

specifies completion status information on return from REJ$. See the definition in Section 2.5.1.

ast

is the address of an optional user-written AST routine to execute after REJ$ completes.

mail

is the address of the connect block sent by the source task and retrieved by GND$. Specify the same address for this and the GND$ *mail* argument. Connect block information is required to reject the connection.

mailen

is the length of the connect block in decimal bytes. The default value is 98. bytes (N.CBL), the short connect block length, not including optional data. For a long connect block, specify 178. bytes (M.CBL), the long connect block length, not including optional data.

out,outlen

define optional user data to send. See the definition in Section 2.5.1.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The task that requested the connection has aborted or requested a disconnect before the rejection could complete.
IE.BAD	Either the temporary link address in the connect block is not valid, or the optional user data buffer exceeds 16. bytes.
IE.IFC	LUN not assigned to NS:.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.
IE.SPC	Invalid buffer argument; either the connect block (*mail*) or the optional user data buffer (*out*) is outside the user task address space, or the connect block is not word aligned.

SND$
(Send Data over a Logical Link)

2.5.14 SND$ — Send Data over a Logical Link

Use:

Issue SND$ from either task to send message data over an established logical link. This macro completes when the other task has actually received the data.

Formats:

label: SND[W]$ *lun*,[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

SND[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*buf*,*buflen*>]

SND[W]$S *lun*,[*efn*],[*status*],[*ast*],<*buf*,*buflen*>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical link over which to send the data. If you initiated the connection, enter the LUN you used in the CON$ macro. If you accepted the connection, enter the LUN you used in the ACC$ macro.

* *efn*

specifies an optional event flag number to set when SND$ completes.

* *status*

specifies completion status information on return from SND$. See the definition in Section 2.5.1, but note this exception:

Word 1: Contains number of bytes sent.

ast

is the address of an optional user-written AST routine to execute after SND$ completes.

buf

is the address of the buffer containing the data to send.

buflen

is the length in bytes (8128. maximum) of the data to send.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.ABO	The logical link was disconnected during I/O operations.
IE.IFC	LUN not assigned to NS:.
IE.NLN	No logical link has been established on the specified LUN.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.SPC	Invalid buffer argument; either the message data buffer (*buf*) is outside the user task address space, or the buffer length (*buflen*) exceeds 8128. bytes.

SPA$
(Specify User AST Routine)

2.5.15 SPA$ — Specify User AST Routine

Use:

Issue SPA$ from either task to specify a user-written AST routine. The AST routine will execute whenever network data arrives in the network data queue.

Issuing SPA$ affects only the data items that subsequently arrive in the queue. However, SPA$ returns a count of all data items in the queue to word 1 of its status block, including those that preceded the macro.

Formats:

label: SPA[W]$ [*lun*],[*efn*],[*status*],[*ast*],<*addr*>

SPA[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*addr*>]

SPA[W]$S [*lun*],[*efn*],[*status*],[*ast*],<*addr*>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

identifies the logical unit number of the network data queue. Use the same LUN you assigned in the OPN$ macro.

* *efn*

specifies an optional event flag number to set when SPA$ completes.

* *status*

specifies completion status information on return from SPA$. See the definition in Section 2.5.1, but note this exception:

Word 1: Contains number of items in network data queue.

ast

is the address of an optional user-written AST routine to execute after SPA$ completes (see the SPA$ programming note that follows).

addr

is the address of a user-written AST routine. This argument is required for executing an AST routine.

You can change the specified AST routine during task execution by specifying a different starting address or eliminate it by zeroing the starting address. When you change the AST during execution, the AST that executes does not push extra information onto the stack, as a normal completion AST does. Therefore, you need not remove anything from the stack.

Error/Completion Codes:

IS.SUC	The macro completed successfully.
IE.IFC	LUN not assigned to NS:.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.PRI	The network is not accessed on the specified LUN.

The following example demonstrates how an application can process all network data at the AST level by using the SPA$ completion AST to simulate the network data AST.

```
MAIN CODE

OPN$S    ...

SPA$S    ... #CMPAST,<SPAAST>        ; Set up SPAAST as the AST entry
                                     ; for network data


;+

; CMPAST - The entry point for completion of the actual SPA directive
;
; SPAAST - The entry point for each arrival of network data
;-
         .ENABLE LSB

CMPAST:  MOV     (SP)+,IOSB          ; Save the SPA$ I/O status block address
         MOV     R0,-(SP)            ; Save R0
         MOV     #IOSB,R0            ; Get the I/O status address

         CMPB    #IS.SUC,(R0)        ; Was the directive successful?
         BNE     20$                 ; If NE, no - just exit from AST
         MOV     2(R0),R0            ; Else, copy current number of ASTs queued
         BEQ     20$                 ; If EQ, nothing queued, exit from AST
         BR      10$                 ; Else, join common code

SPAAST:  MOV     R0,-(SP)            ; Save R0
         MOV     #1,R0               ; Set the network data queue count to one
10$:     GNDW$S  ,,,,,#GNDSB         ; Get the network data item
         BCS     20$                 ; If CS, directive failed
         CMPB    #IS.SUC,GNDSB       ; Was the directive successful ?
         BNE     20$                 ; If NE, no - exit from AST
                                     ;
         ...                         ; ... do some processing ...

         SOB     R0,10$              ; Continue until
20$:     MOV     (SP),R0             ; Restore R0
         ASTX$S                      ; Exit from AST

         .DSABL  LSB
```

XMI$
(Send Interrupt Message)

2.5.16 XMI$ — Send Interrupt Message

Use:

Issue XMI$ from either task to send an interrupt message over an established logical link. XMI$ places the message on the target task's network data queue. The target task must issue GND$ to retrieve the message before you can issue another XMI$ on the same logical link. Note that XMI$ may complete before the target task issues a GND$ to retrieve the interrupt message.

Formats:

label: XMI[W]$ *lun*,[*efn*],[*status*],[*ast*],<*int*,*intlen*>

XMI[W]$E *label*,[*lun*],[*efn*],[*status*],[*ast*][,<*int*,*intlen*>]

XMI[W]$S *lun*,[*efn*],[*status*],[*ast*],<*int*,*intlen*>

Arguments:

label

specifies the location of the argument block. See the definition in Section 2.5.1.

lun

specifies the logical link over which to send the interrupt message. If you initiated the connection, enter the LUN you used in the CON$ macro. If you accepted the connection, enter the LUN you used in the ACC$ macro.

* *efn*

specifies an optional event flag number to set when XMI$ completes.

* *status*

specifies completion status information on return from XMI$. See the definition in Section 2.5.1, but note this exception:

Word 1: Contains number of bytes sent in message.

ast

is the address of an optional user-written AST routine to execute after XMI$ completes.

int

is the address of the buffer that contains the 1- to 16.-byte interrupt message to send.

intlen

is the length in decimal bytes of the message to send

Error/Completion Codes:

IS.SUC	The interrupt message has been transmitted successfully. This code does not ensure that GND$ retrieved the message.
IE.ABO	The logical link was disconnected during I/O operations.
IE.BAD	The interrupt message exceeds 16. bytes.
IE.IFC	LUN not assigned to NS:.
IE.NLN	No logical link has been established on the specified LUN.
IE.NNT	The issuing task is not a network task; OPN$ did not execute successfully.
IE.SPC	Invalid buffer argument; the interrupt message buffer (*int*) is outside the user task address space.
IE.WLK	An interrupt message was transmitted before a previous interrupt message had been received by the target task.

2.5.17 MACRO–11 Intertask Communication Programming Examples

The following MACRO–11 programs are cooperating programs to run on different nodes in the network. The transmitting program, SEN10, sends messages to the receiving program, REC10.

These programming examples are included in your tape or disk kit.

2.5.17.1 Transmit Example

The SEN10 program transmits an interrupt message and 10 data messages to the cooperating REC10 program.

```
        .TITLE  SEN10
;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
; This program prompts a user for the text of a message to transmit to
; the remote receiving task REC10.  It sends that message as an interrupt
; message and sends 10 data messages with the format "This is message n"
; to REC10.
;
; To assemble, use the following command string:
;
;       MAC SEN10,SEN10/-SP=IN:[100,10]NETLIB/ML,IN:[200,200]SEN10
;
; To task build, use the following command string:
;
;       TKB SEN10,SEN10/-SP=SEN10,IN:[130,10]NETLIB/LB
;
; Note: The IN: device must be the DECnet distribution device
;       after the PREGEN (if any) has been performed.
;
;*************************************************************
        .MCALL  OPNW$S,CONW$S,SNDW$S,CONB$$,ALUN$C,QIOW$C
        .MCALL  EXIT$S,MRKT$C,WTSE$C,CLEF$C,SETF$C,QIO$C
        .MCALL  DSCW$S,XMIW$S,ASTX$S
;
; Data area
;
MESN:   .ASCII  /This is message /      ; Message to transmit
NUM:    .ASCII  /0/                     ; Message number
NN=.-MESN
PRMPT:  .ASCII  /MSG:/                  ; Prompt for interrupt message
        .EVEN
IOSTN:  .BLKW   2                       ; Completion status for network
BUFF:   .BLKB   16.                     ; Interrupt message buffer
IOSTB:  .BLKW   2                       ; Completion status for buffer
CNT:    .WORD   0                       ; Number of chars in interrupt message
```

```
ERRCNT: .WORD   0                       ; Error count
IOSB:   .BLKW   1                       ; I/O status
;
        .EVEN
CONBL:  CONB$$  TACOMA,0,1,<REC10>      ; Connect request block
;
;  CODE
;
        .EVEN
START:  CLR     ERRCNT                  ; Initialize error count to zero
        CLEF$C  5                       ; Clear event flag used to make sure
                                        ; Interrupt message accepted prior
                                        ;   to exit
        MOVB    #60,NUM                 ; Initialize message num to zero
        ALUN$C  1,NS                    ; Assign LUN 1 for network data queue
        ALUN$C  2,NS                    ; Assign LUN 2 for logical link
        OPNW$S  #1,#1,#IOSTN            ; Create the network data queue
        TSTB    IOSTN                   ; Test for errors
        BGT     OK1
        JMP     ERR1
OK1:    CONW$S  #2,#2,#IOSTN,,<#CONBL>  ; Create logical link to "REC10"
        TSTB    IOSTN                   ; Test for errors
        BLE     ERR2
        QIO$C   IO.RPR,5,,,IOSTB,TRMAST,<BUFF,16.,,PRMPT,4> ; Accept
                                        ; interrupt message from terminal
                                        ; (use AST)[16 char max]
        TST     $DSW                    ; Test for errors
        BLT     ERR3
        MOV     #10.,R0                 ; Set loop counter to 10
LOOP:   SNDW$S  #2,#2,#IOSTN, ,<#MESN,#NN> ; Send message
        TSTB    IOSTN                   ; Test for errors
        BLE     ERR4
        INCB    NUM                     ; Update message number
        SOB     R0,LOOP                 ; Loop if more to send
;
        WTSE$C  5                       ; Make sure terminal message was
                                        ; entered before exiting
;
        DSCW$S  #2,#2,#IOSTN            ; Disconnect network
;
        EXIT$S                          ; Exit
;
;  Terminal AST routine
;
TRMAST: MOV     (SP)+,IOSB              ; Pop stack
        MOV     IOSTB+2,CNT             ; Obtain number of characters
        XMIW$S  #2,#3,#IOSTN, ,<#BUFF,CNT>; Transmit interrupt message
                                        ; (Note use of EF 3 instead of
                                        ;  EF 2 - avoid competition)
        TSTB    IOSTN                   ; Test for errors
        BLE     ERR5
        SETF$C  5                       ; Set event flag to indicate that
                                        ;  interrupt message sent
        ASTX$S                          ; AST exit
;
; Error handling - a sample debugging technique
;
ERR5:   INC     ERRCNT                  ; Determine
ERR4:   INC     ERRCNT                  ;  which
ERR3:   INC     ERRCNT                  ;  error
ERR2:   INC     ERRCNT                  ;  occurred
```

```
ERR1:   INC     ERRCNT
        MOV     ERRCNT,R1               ; R1 contains the error number
        MOV     $DSW,R2                 ; R2 contains the Directive Status Word
        MOV     IOSTN,R3                ; R3 contains the first I/O status word
        MOV     IOSTN+2,R4              ; R4 contains the 2nd I/O status word
        IOT                             ; Abort - dump the registers
;
;
;
        .END    START
```

2.5.17.2 Receive Example

Each time REC10 receives a message from the cooperating program SEN10, it displays THIS IS MESSAGE *n* on TI:. This is followed by the actual message, which arrives as an interrupt message.

```
        .TITLE  REC10

;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
;  This example receives short messages from the sender task "SND10,"
;  prints the messages on TI:, disconnects, and exits gracefully.
;
;  To assemble, use the following command string:
;
;       MAC REC10,REC10/-SP=IN:[130,10]NETLIB/ML,IN:[200,200]REC10
;
;  To task build, use the following command string:
;
;       TKB REC10,REC10/-SP=REC10,IN:[130,10]NETLIB/LB
;
;    Note: The IN: device must be the DECnet distribution device
;          after the PREGEN (if any) has been performed.
;

        .MCALL  OPNW$S,SPAW$S,RECW$S,GNDW$S,ACCW$S,CLSW$S,NETDF$
        .MCALL  QIOW$S,ALUN$C,CLEF$C,WTSE$C,SETF$C,ASTX$S,EXIT$S
        NETDF$
;
; Data area
;
BUF1:   .BLKB   25.                     ; Buffer for user messages
        .EVEN
BUF2:   .BLKB   N.CBL                   ; Buffer for network messages
IOST:   .BLKW   2                       ; Completion status for network
IOST1:  .BLKW   2                       ; Comp. stat. for Get Net Data
IOST2:  .BLKW   2                       ; Comp. stat. for Accept Connect
IOSB:   .BLKW   1                       ; I/O status
ERRCNT: .WORD   0                       ; Error count
CNT:    .WORD   0                       ; User message char count
CNTB:   .BLKB   2                       ; Interrupt message char count
FLAG:   .WORD   0                       ; Disconnect flag
        .EVEN
```

```
; Code
;
START:  CLR     ERRCNT                  ; Initialize error count to zero
;       clef$c  10.                     ; Clear event flag used to make
                                        ;   sure connect has occurred
        ALUN$C  1,NS                    ; Assign LUN 1 for network data queue
        ALUN$C  2,NS                    ; Assign LUN 2 for logical link
        OPNW$S  #1,#1,#IOST             ; Create the network data queue
        TSTB    IOST                    ; Test for errors
        BLE     ERR1
        SPAW$S  #1,#1,#IOST,#CMPAST,<#NETAST> ; Specify AST handling
        TSTB    IOST                    ; Test for errors
        BLE     ERR2
        WTSE$C  10.                     ; Wait to make sure connect occurred

LOOP:   RECW$S  #2,#2,#IOST,,<#BUF1,#25.>; Receive up to 25 chars
        TSTB    IOST                    ; Test for errors
        BLE     ERR3
        MOV     IOST+2,CNT              ; Obtain character count
        QIOW$S  #IO.WLB,#5,#5,,,,<#BUF1,CNT,#40>; Type message on terminal
        TST     FLAG                    ; Has disconnect occurred?
        BEQ     LOOP                    ; No, post another receive
        CLSW$S  #1,#1,#IOST2            ; Close network
        TSTB    IOST2                   ; Test for errors
        BLE     ERR5
        EXIT$S                          ; Program exit
        BR      LOOP
;
; Error handling - a sample debugging technique
;
ERR6:   INC     ERRCNT
ERR5:   INC     ERRCNT
ERR4:   INC     ERRCNT
ERR3:   INC     ERRCNT
ERR2:   INC     ERRCNT
ERR1:   INC     ERRCNT
        MOV     ERRCNT,R1               ; R1 = Error number
        MOV     $DSW,R2                 ; R2 = Directive Status Word
        MOV     IOST,R3                 ; R3 = I/O status block (1st word)
        MOV     IOST+2,R4               ; R4 = I/O status block (2nd word)
        IOT                             ; Abort - dump registers
;
; AST handling for data in network data queue
;
CMPAST: MOV     (SP)+,IOSB              ; Save SPA$ I/O status block addr
        MOV     R0,-(SP)                ; Save R0
        MOV     IOSB,R0                 ; Get I/O status block address
        CMPB    #IS.SUC,(R0)            ; Successful?
        BEQ     OKA
        JMP     OUT
OKA:    MOV     2(R0),R0                ; Get current network data count
        BNE     OKB
        JMP     OUT
OKB:    BR      GET
NETAST: MOV     R0,-(SP)                ; Save R0
        MOV     #1,R0                   ; Set network data count to 1
GET:    GNDW$S  #1,#1,#IOST1,,<#BUF2,#N.CBL> ; Get network data
        BCS     OUT                     ; Carry bit set - error
        CMPB    #IS.SUC,IOST1           ; Successful?
        BNE     OUT
        CMPB    #NT.CON,IOST1+1         ; Check if connect request
```

```
        BNE     OTHER
        ACCW$S  #2,#2,#IOST2,,<#BUF2>   ; Accept connection
        TSTB    IOST2                   ; Test for errors
        BLE     ERR4
        SETF$C  10.                     ; Set event flag to indicate
                                        ; connect occurred
        BR      NEXT
OTHER:  CMPB    #NT.DSC,IOST1+1         ; Check if disconnect request
        BNE     OTHR2
        MOV     #1,FLAG                 ; Set disconnect flag
        BR      NEXT                    ; Go back to main routine
;
;
OTHR2:  CMPB    #NT.INT,IOST1+1         ; Check if interrupt message
        BEQ     OKC
        JMP     ERR6                    ; Not a expected command
OKC:    MOVB    IOST1+2,CNTB            ; Obtain character count
        QIOW$S  #IO.WLB,#5,#3, , , ,<#BUF2,CNTB,#40> ; Type interrupt message
                                                     ; (Note use of EF 3
                                                     ;  instead of EF 5)
NEXT:   NOP
        DEC     R0                      ; Check if more data
        BEQ     OUT
        JMP     GET
OUT:    MOV     (SP)+,R0                ; Restore R0
        ASTX$S                          ; AST exit
;
;
;
        .END    START
```

3

FORTRAN, COBOL, and BASIC–PLUS–2 Programming Facilities

DECnet–RSX has three types of network subroutines:

- Intertask communication calls
- Remote file access calls
- FORTRAN task control calls

This chapter lists the calls that perform these subroutines in alphabetical order. The description for each call includes its use, formats, argument definitions, and error/completion codes. All references to FORTRAN pertain to both FORTRAN IV and FORTRAN 77. All references to BASIC pertain to BASIC–PLUS–2. Before issuing these calls, read Chapter 1.

3.1 Building a DECnet–RSX Task

When a FORTRAN, COBOL, or BASIC task uses any DECnet–RSX facility, that task must be linked to the library [1,1] NETFOR.OLB. For example, a COBOL task named FILES can be built under RSX–11M with the following task builder command string:

```
FILES,FILES=FILES,LB:[1,1]NETFOR/LB,LB:[1,1]COBLIB/LB
/
TASK=FILES
PAR=GEN
ACTFIL=2
//
```

You need not assign logical unit numbers (LUNs) for calls to the network (NS:) at task build time or in your program. The OPNNT[W], CONNT[W],and ACCNT[W] calls assign the LUNs to the network at run time. Assigning a LUN to the network at task build time or in your program will not have an adverse effect on the execution of the program. Be sure that the LUNs you specify in these three calls are used only for network activity while assigned to NS:.

3.2 Establishing a Network Task

The first DECnet call you issue must be an open call. To access the network, issue one of the following open calls:

OPNNT	Establishes your task as an active network task and creates a network data queue for it.
OPNNTW	Performs the same function as OPNNT, but suspends further task execution until the call completes.

After opening the task to the network, you can establish a logical link by issuing calls described in this chapter.

To terminate network operations for a task, issue one of the following closing calls:

CLSNT	Terminates a task's network activity, aborts its established logical links, and frees all its network logical unit numbers.
CLSNTW	Performs the same function as CLSNT, but suspends further task execution until the call completes.

3.3 Examining I/O Status Blocks

All calls in this chapter let you include an argument that specifies the address of a status block. This address contains completion status information when the call completes.

The status block address is recommended but optional for intertask communication and task control calls. It is required for remote file access calls.

Status blocks are either 1- or 2-element integer arrays/strings. The BACC, BACCL, BFMT0, and BFMT1 calls use one-element arrays/strings. In these arrays/strings, a return of –1 indicates that you supplied valid arguments; 0 indicates invalid arguments.

Other calls use 2-element arrays/strings. In these arrays/strings, the first status word contains an error/completion code for the call, as follows:

- A positive value indicates that the call executed successfully.
- A negative value indicates that the call did not execute properly.
- A null value (0) indicates the call has not yet completed.

Examine the value of the returned error/completion code to determine why a call failed. Appendix E gives a complete list of error/completion codes for intertask calls and task control calls.

The contents of the second status word differ according to the call you issue. Each call therefore defines the contents of the second status word.

3.4 Using Event Flags

The network file access routines (NFARs) require the exclusive use of two event flags. The default event flags are 17 (.TREF) and 18 (.RCEF). You can choose to override these defaults by issuing the following commands in the task builder command file:

```
GBLDEF=.TREF:value
GBLDEF=.RCEF:value
```

The *value* variable is a decimal integer from 1 to 64. (33. through 64. are global flags).

3.5 Specifying Connect Block Options

As Chapter 1 described, a source task builds a connect block before issuing a connect request. This outgoing connect block contains information about the connect request's target node and task. It can also specify explicit access control information that gives the source task access to an account on the target node. Before network software sends the connect block to the target task, it adds information about the source task or user. If you have an RSX-11M-PLUS or Micro/RSX system with outgoing proxy enabled, network software also adds proxy information (see Section 3.6). At the target node, the target task retrieves the incoming connect block from the network data queue.

Your task can use either long or short connect blocks. Using long connect blocks lets your task support user IDs, passwords, and accounts of 39. characters each.

Using short connect blocks lets your task support user IDs of up to 16. characters, passwords of up to 8. characters, and accounts of up to 16. characters.

For greatest flexibility, use long connect blocks when writing a new task. However, you can continue to use an existing task that uses short connect blocks without modifying the task. If you change an existing task to use long connect blocks, note the added buffer space requirements. Also note that if the task uses proxy access, you need not supply values for the access control information fields.

The connect block size that you choose affects the following intertask communication calls:

Call	Connect Block Option
OPNNT	Include the *mbxflg* argument to specify a long connect block.
BACC	Call BACC to build a short connect block.
BACCL	Call BACCL to build a long connect block.
CONNT	Specify the size appropriate to your connect block type in the *tgtblk* argument.
ACCNT GNDNT REJ$	Use the *mailbuf* argument to reference the appropriately-sized buffer.

The connect block size also affects the following FORTRAN task control calls:

Call	Connect Block Option
ABONCW RUNCW	Specify the appropriate length for the *passwd* argument.
BACUSR	Call BACUSR for a short account and user ID information area.
BACUSL	Call BACUSL for a long account and user ID information area.

When access verification for your task is on, your node's network software verifies access rights and removes the access control information before passing an incoming connect block to your task. For information on enabling verification for a task, refer to the *DECnet–RSX Guide to Network Management Utilities*.

3.5.1 Receiving Connect Block Information

You specify the task's connect block type by including or omitting the *mbxflg* (mailbox flag) argument to the OPNNT (access the network) call. If you specify the *mbxflg* argument, network software uses long connect blocks when passing access control information to your task.

If the source and target tasks use the same connect block size, incoming connect block fields map directly to receiving fields. Communicating tasks need not use the same connect block size, however. In the GNDNT call, which retrieves the connect block from the network data queue, you specify a buffer to hold the retrieved information. The buffer that you allocate may or may not equal the incoming connect block, but in writing incoming data to your buffer, network software always uses the offsets appropriate to the connect block size that your OPNNT call specified.

You receive all information if the source task sends the same size connect block that you receive, or if you receive long connect blocks and the source task sends a short connect block. However, if you receive short connect blocks and the source task sends a long connect block, you may lose some information. Network software writes the received information into the appropriate field if the information fits. Information that does not fit into the receiving field causes a data overrun error and is lost.

You can choose to allocate a receiving buffer that is smaller or larger than the expected connect block. For example, you might allocate a smaller buffer to exclude all but the initial fields, or allocate a larger buffer to receive optional user data. The GNDNT call description describes what happens when the task receives access control information that is smaller or larger than expected.

3.6 Using Access Control Information

An outgoing connect request sends the target node information in order to gain access to the target node. You can specify the access control information and/or the network software can supply proxy information. Proxy access is available only with RSX–11M–PLUS or Micro/RSX.

When you supply explicit access control information for the connect request, you specify a user ID, password, and, optionally, an account number. These identify the target account on the remote node. You specify the explicit access control information by calling BACC or BACCL. These calls build the connect block's access control information area. When the target system receives the connect

request, it grants access according to what you specified. For information on which access control arguments the target system requires, refer to user documentation for that system; DECnet–RSX nodes require the user ID and password. For more information on explicit access control information, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.

You can supply access control information with intertask communication calls, and remote file access calls.

- **Intertask communication calls.** A source task supplies access control information in the BACC or BACCL call.

- **Remote file access calls.** A source task supplies access control information in the *ident* argument.

You can also define an alias node name that includes explicit access control information. An alias node name is a user-assigned logical name for a network node. When a source task user defines an alias node name, the task can omit the access control information; access control information associated with the alias is used automatically. The *DECnet–RSX Network Management Concepts and Procedures* manual has more information on creating and using alias node names.

Proxy access, in contrast, eliminates the need to send passwords across the network. The network managers on both nodes must set up the environment for using proxy. Once your network manager enables outgoing proxy, your node automatically sends proxy information with all outgoing connect requests. Proxy information is the user ID under which the source task is executing. If incoming proxy is enabled on the target node, the system grants access according to the proxy information and source node name. For information on how a target system verifies proxy access, refer to the *DECnet–RSX Guide to Network Management Utilities*.

If an incoming connect request contains both types of access control information, the target system uses the explicit information, and not the proxy information, to verify access.

3.7 Conventions Used in This Chapter

The following notation conventions are used in the call and argument descriptions and examples for intertask communication, remote file access, and task control calls in this chapter:

asterisk *

flags arguments relating to arrays/character strings that you must check for information after the call completes. For example, the *status* argument specifies an array/data item that stores completion status information when the call completes.

UPPERCASE

indicates characters to type exactly as shown.

lowercase italics

indicate variables for which you specify or the system supplies the actual values.

commas, periods, parentheses ( )

must be typed where shown as part of the call format. Even if you omit an argument, include the comma that delineates its field unless no other arguments follow.

FORTRAN Example:

Basic call format:

CALL BACC ([*status*],*tgtblk*,[*usersz*,*user*],
[*passwdsz*,*passwd*][,*accnosz*,*accno*])

Sample call:

```
CALL BACC (,tgtblk,,passwdsz,passwd)
```

The example omits arguments for *status*, *usersz*, *user*, *accnosz*, and *accno*. Commas delineate the fields for the first three missing arguments, but are unnecessary for the two arguments dropped at the end of the call.

numbers

represent octal numbers in calls and examples unless followed by a decimal point.

Example:

A 1- to 72.-element character string

square brackets [ ]

enclose optional data. You must specify any argument not enclosed by brackets. Do not type the brackets when you code a call.

In COBOL and BASIC, you can omit an optional argument only if you also omit all trailing arguments. However, you can enter 0 for an optional argument that you want to omit, but that has trailing arguments you want to include.

COBOL Example:

Basic call format:

CALL "CONNT" USING *lun*,[*status*],*tgtblk*,
[*outsize*,*outmessage*],
[*insize*,*inmessage*].

This call includes three categories of optional data:

- *status*

  is optional but cannot be omitted because it is followed by a required argument, *tgtblk*. You can enter 0 for *status* to prevent the return of status information for the call.

- *outsize*,*outmessage*

  are paired optional arguments that you can omit only if you also omit the trailing arguments, *insize* and *inmessage*. To omit *outsize* and *outmessage*, but include the arguments that follow, enter null arguments (0) for *outsize* and *outmessage*.

- *insize*,*inmessage*

  are paired optional arguments that you can omit without specifying null values since there are no trailing arguments.

Sample call:

```
CALL "CONNT" USING lun,status,tgtblk,0,0,insize,inmessage.
```

This call specifies *status* and *insize*,*inmessage*, while omitting *outsize*,*outmessage* by specifying null values for these optional arguments.

3.8 Intertask Communication

This section contains descriptions and usage guidelines for the intertask communication calls that Table 3–1 lists alphabetically.

Read the preceding material in this chapter before using these calls. If you are unfamiliar with network intertask communication concepts, also read Chapter 1 carefully.

Table 3–1: Intertask Communication Calls

Call	Function
ABTNT	Abort a logical link
ACCNT	Accept a logical link connect request
BACC	Build access control information area (short)
BACCL	Build access control information area (long)
BFMT0	Build a format 0 destination descriptor
BFMT1	Build a format 1 destination descriptor
CLSNT	End a task's network operations
CONNT	Request a logical link connection
DSCNT	Disconnect a logical link
GLNNT	Get local node information
GNDNT	Get data from network data queue
OPNNT	Access the network
RECNT	Receive data over a logical link
REJNT	Reject logical link connect request
SNDNT	Send data over a logical link
WAITNT	Suspend the calling task
XMINT	Send interrupt message over a logical link

Each call description includes the format for each language. The generic formats for each language are:

FORTRAN: CALL *xxxxx* (*arguments*)

COBOL: CALL "*xxxxx*" USING *arguments*.

BASIC: CALL *xxxxx* BY REF (*arguments*)

3.8.1 Common Argument Definitions

This section defines the common arguments for intertask communication calls. A general group defines arguments common to all languages and three individual groups define arguments specific to FORTRAN, COBOL, and BASIC–PLUS–2.

GENERAL

- *outsize,outmessage*

  define optional user data to send with certain calls. These are paired optional arguments; use both or omit both.

EXCEPTION

You cannot omit *outsize,outmessage* in the CONNT call in COBOL and BASIC unless you also omit the *insize,inmessage* arguments. To include *insize,inmessage* without specifying *outsize,outmessage*, enter a null value (0) for both *outsize* and *outmessage*. (See the example under the discussion of square brackets in Section 3.7.)

outsize specifies the length in bytes/characters of the optional user data you can send on some operations. It must be an integer variable or constant.

outmessage specifies the array/string containing the user data to send. This is a 1- to 16.-element byte array for FORTRAN or a 1- to 16.-element numeric data item/character string for COBOL or BASIC.

FORTRAN

- References to integers imply single-precision integer values.

- *status*

  specifies an array containing completion status information on return from the call. If specified, this 2-element single-precision integer array contains the following values when the call completes:

  status(1) returns an error/completion code. Refer to the descriptions of individual calls for a list of the possible codes.

  status(2) returns a directive error code if *status*(1) returns a value of −40. Otherwise, *status*(2) contains 0.

- *tgtblk*

  specifies an array where the BACC or BACCL call builds the explicit access control information area and the BFMT0 or BFMT1 call builds the destination descriptor. A short connect block requires 72. bytes; a long connect block requires 152. bytes. The array must start on an even byte (word) boundary. A CONNT call passes this array to the target task.

COBOL

- For a COBOL task using the DECnet interface, logical unit number 1 is a reserved number and should never be assigned for a *lun*.

- *status*

  specifies an elementary numeric data item containing completion status information on return from the call. If specified, this elementary numeric data item contains the following values when the call completes:

  status(1) returns an error/completion code. Refer to the descriptions of individual calls for a list of the possible codes.

  status(2) returns a directive error code if *status*(1) returns a value of −40. Otherwise, *status*(2) contains 0.

  You cannot omit *status* if there are trailing arguments, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

- *tgtblk*

  specifies a numeric data item that specifies the area where the BACC or BACCL call builds the explicit access control information area and the BFMT0 or BFMT1 call builds the destination descriptor. A short connect block requires 72. bytes; a long connect block requires 152. bytes. A CONNT call passes the explicit access control and destination descriptor information to the target task.

BASIC-PLUS-2

- *status*%( )

  specifies an array containing completion status information on return from the call. If specified, this 2-element integer array contains the following values when the call completes:

  status%(0) returns an error/completion code. Refer to the descriptions of individual calls for a list of the possible codes.

  status%(1) returns a directive error code if *status*%(0) returns a value of −40. Otherwise, *status*%(1) contains 0.

  You cannot omit *status* if there are trailing arguments, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

- *tgtblk*$

  specifies a character string that specifies the area where the BACC or BACCL call builds the explicit access control information and the BFMT0 or BFMT1 call builds the destination descriptor. A CONNT call passes this string to the target task. A short connect block requires 72. bytes; a long connect block requires 152. bytes. To allocate space for *tgtblk*$, use the STRING function:

  ```
  tgtblk$=STRING$(152,0)
  ```

ABTNT
(Abort Logical Link)

3.8.2 ABTNT — Abort Logical Link

Use:

Call ABTNT from either task to abort a logical link. ABTNT immediately aborts all pending transmits and receives, disconnects the link, and frees the LUNassigned to the logical link. When you call ABTNT, you can send 1 to 16. bytes/characters of user data to the task from which you are disconnecting (see the *outsize, outmessage* arguments).

Formats:

FORTRAN: CALL ABTNT[W] (*lun*,[*status*][,*outsize*,*outmessage*])

COBOL: CALL "ABTNT[W]" USING *lun*,[*status*][,*outsize*,*outmessage*].

BASIC: CALL ABTNT[W] BY REF (*lun*%,[*status*%( )]
[,*outsize*%,*outmessage*$])

Arguments:

lun

identifies the logical link to abort. This value must be an integer variable or constant. If you initiated the connection, enter the LUN you used in the CONNT call. If you accepted the connection, enter the LUN you used in the ACCNT call.

* *status*

specifies completion status information on return from ABTNT. See the definition for your language in Section 3.8.1.

outsize,*outmessage*

define optional user data to send. See the definition in Section 3.8.1.

Error/Completion Codes:

1	The call completed successfully.
−2	No logical link has been established on the specified LUN.
−9	The task is not a network task; OPNNT did not execute successfully.
−13	You are using an invalid buffer; the optional *outmessage* buffer is outside the user task address space.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

ACCNT
(Accept Logical Link Connect Request)

3.8.3 ACCNT — Accept Logical Link Connect Request

Use:

Call ACCNT from the target task to establish a logical link with the source task. When you call ACCNT, you can send 1 to 16. bytes/characters of user data to the source task (see the *outsize*, *outmessage* arguments).

Formats:

FORTRAN: CALL ACCNT[W](*lun*,[*status*],*mailbuf*,[*outsize*,*outmessage*])

COBOL: CALL "ACCNT[W]" USING *lun*,[*status*],*mailbuf*
[,*outsize*,*outmessage*]

BASIC: CALL ACCNT[W] BY REF (*lun*%,[*status*%( )],*mailbuf*$
[,*outsize*%,*outmessage*$])

Arguments:

lun

assigns the logical link number. This value must be an integer variable or constant. Use this LUN when referring to this logical link in any succeeding RECNT, SNDNT, XMINT, ABTNT, or DSCNT call.

* *status*

specifies completion status information on return from ACCNT. See the definition for your language in Section 3.8.1.

mailbuf

specifies a 1- to *n*-element array/string that contains the connect block. In FORTRAN, this array must start on an even byte (word) boundary. For more information, see Table 3–2 and the description of *mailbuf* under GNDNT (Section 3.8.12).

outsize,*outmessage*

define optional user data to send. See the definition in Section 3.8.1.

Error/Completion Codes:

1	The call completed successfully.
−1	System resources needed for the logical link are unavailable.
−3	The task that requested the connection has aborted or requested a disconnect before the connection could complete.
−5	The temporary link address in the mail buffer is not valid.
−8	A logical link has already been established on the specified LUN.
−9	The task is not a network task; OPNNT did not execute successfully.
−13	You are using an invalid buffer; the *mailbuf* or *outmessage* buffer is outside the user task address space, or (for FORTRAN) *mailbuf* is not word aligned.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

BACC
(Build Access Control Information Area (Short))

3.8.4 BACC — Build Access Control Information Area (Short)

Use:

Call BACC from the source task to build an area for explicit access control information for the outgoing connect block. BACC supports 16.-character user IDs, 8.-character passwords, and 16.-character accounts.

Explicit access control information arguments define your access rights at the remote node or process. The target system verifies access control information according to its system conventions. If the target node is equipped to verify the information, it does so before passing the CONNT call to the target task. For more information on access control verification, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.

If you have defined an alias node that includes the explicit access control information, or if you use proxy access, you need not call BACC.

Formats:

FORTRAN: CALL BACC ([*status*],*tgtblk*,[*usersz*,*user*],
[*passwdsz*,*passwd*][,*accnosz*,*accno*])

COBOL: CALL "BACC" USING [*status*],*tgtblk*,[*usersz*,*user*],
[*passwdsz*,*passwd*][,*accnosz*,*accno*].

BASIC: CALL BACC BY REF ([*status*%],*tgtblk*$,[*usersz*%,*user*$],
[*passwdsz*%,*passwd*$][,*accnosz*%,*accno*$])

Arguments:

* *status*

  specifies an integer variable containing completion status information on return from BACC. On return, the variable is set to –1 if the BACC call completed successfully or to 0 if there was an invalid BACC argument.

  In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

* *tgtblk*

  specifies an array/string in which to build the explicit access control information area. See the definition for your language in Section 3.8.1.

  usersz,user

  specify the user ID. These are paired optional arguments; use both or omit both (FORTRAN) or enter 0 for both (COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets) in Section 3.7.

  usersz specifies the user ID length in bytes/characters. This field is an integer variable or constant.

  user specifies the 1- to 16.-element array/string containing the user ID.

  passwdsz,passwd

  specify the password that determines your access at the remote node. These are paired optional arguments; use both or omit both (FORTRAN) or enter 0 for both (COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets) in Section 3.7.

  passwdsz specifies the password length in bytes/characters. This field is an integer variable or constant.

  passwd specifies a 1- to 8.-element array/string containing the password.

accnosz,accno

specify the account number. These are paired optional arguments; use both or omit both.

accnosz specifies the account number length in bytes/characters. Do not use this argument for RSX target systems. This field is an integer variable or constant.

accno specifies a 1- to 16.-element array/string containing the account number.

Connect Block Offsets:

Length in Decimal Bytes/Characters	Destination Descriptor
26.	Built by BFMT0 or BFMT1 call
	Access Control
2.	User ID length (16. bytes/characters or less)
16.	User ID
2.	Password length (8. bytes/characters or less)
8.	Password
2.	Account number length (16. bytes/characters or less)
16.	Account number

BACCL
(Build Access Control Information Area (Long))

3.8.5 BACCL — Build Access Control Information Area (Long)

Use:

Call BACCL from the source task to build an area for explicit access control information for the outgoing connect block. BACCL supports 39.-character user IDs, passwords, and accounts.

The explicit access control information arguments define an account at the remote node. The target system verifies access control information according to its system conventions. If the target node is equipped to verify the information, it does so before passing the CONNT call to the target task.

If you have defined an alias node name that includes the access control information, or if you use proxy access, you need not call BACCL. For more information on access control verification, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on using aliases, refer to the *DECnet–RSX Guide to User Utilities* or the *DECnet–RSX Network Management Concepts and Procedures* manual. For more information on proxy access, refer to the *DECnet–RSX Guide to Network Management Utilities*.

Formats:

FORTRAN: CALL BACCL ([*status*],*tgtblk*,[*usersz*,*user*],
[*passwdsz*,*passwd*][,*accnosz*,*accno*])

COBOL: CALL "BACCL" USING [*status*],*tgtblk*,[*usersz*,*user*],
[*passwdsz*,*passwd*][,*accnosz*,*accno*].

BASIC: CALL BACCL BY REF ([*status*%],*tgtblk*$,[*usersz*%,*user*$],
[*passwdsz*%,*passwd*$][,*accnosz*%,*accno*$])

Arguments:

* *status*

  specifies an integer variable containing completion status information on return from BACCL. On return, the variable is set to –1 if the BACCL call completed successfully or to 0 if there was an invalid BACCL argument.

  In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

* *tgtblk*

  specifies an array/string in which to build the explicit access control information area. See the definition for your language in Section 3.8.1.

  usersz,user

  specify the user ID. These are paired optional arguments; use both or omit both (FORTRAN) or enter 0 for both (COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets) in Section 3.7.

  usersz specifies the user ID length in bytes/characters. This field is an integer variable or constant.

  user specifies the 1- to 39.-element array/string containing the user ID.

  passwdsz,passwd

  specify the password that determines your access at the remote node. These are paired optional arguments; use both or omit both (FORTRAN) or enter 0 for both (COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets) in Section 3.7.

  passwdsz specifies the password length in bytes/characters. This field is an integer variable or constant.

  passwd specifies a 1- to 39.-element array/string containing the password.

accnosz,accno

specify the account number. These are paired optional arguments; use both or omit both.

accnosz specifies the account number length in bytes/characters. Do not use this argument for RSX target systems. This field is an integer variable or constant.

accno specifies a 1- to 39.-element array/string containing the account number.

Connect Block Offsets:

Length in Decimal Bytes/Characters	Destination Descriptor
26.	Built by BFMT0 or BFMT1 call
	Access Control
2.	User ID length (39. bytes/characters or less)
39.	User ID
2.	Password length (39. bytes/characters or less)
39.	Password
2.	Account number length (39. bytes/characters or less)
39.	Account number

BFMT0
(Build a Format 0 Destination Descriptor)

3.8.6 BFMT0 — Build a Format 0 Destination Descriptor

Use:

Call BFMT0 from the source task to build a format 0 destination descriptor for the connect block. Use a format 0 descriptor to connect only to a target task that requires specification of an object type.

Object types group DECnet programs according to function and are identified throughout the network by object type codes (see Appendix B). For example, the TLK server task, LSN, has an object type code 016 (decimal). Any other program that provides the same function on another DECnet system also has object type code 016 (decimal), regardless of its name.

Formats:

FORTRAN: CALL BFMT0 ([*status*],*tgtblk*,*ndsz*,*ndname*,*objtype*)

COBOL: CALL "BFMT0" USING [*status*],*tgtblk*,*ndsz*,*ndname*,*objtype*.

BASIC: CALL BFMT0 BY REF ([*status*%],*tgtblk*$,*ndsz*%,*ndname*$, *objtype*%)

Arguments:

* *status*

  specifies an integer variable containing completion status information on return from BFMT0. On return, the variable is set to .TRUE. (for FORTRAN) or to −1 (for COBOL and BASIC) if the BFMT0 call completed successfully. It is set to .FALSE. (for FORTRAN) or to 0 (for COBOL and BASIC) if there was an invalid BFMT0 argument.

  In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

* *tgtblk*

specifies an array/string in which to build the destination descriptor. See the definition for your language in Section 3.8.1.

ndsz

specifies the node name length in bytes/characters. This field must be an integer variable or constant.

ndname

specifies a 1- to 6-element array/string containing the name of the target node.

objtype

is the target task's object type. The *objtyp* argument is an integer variable or constant. The object type for a named object is 0. The object type for a numbered object is in the range 1 to 127. for a DECnet task or 128. to 255. for a user task. Refer to Appendix B for a list of object type codes.

Privileged users can define their own object types. For more information, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.

Connect Block Offsets:

Length in Decimal Bytes/Characters	Destination Descriptor
6.	Destination node name with trailing blanks
1.	Descriptor format type (0 for BFMT0)
1.	Destination object type (1 to 255.)
	Descriptor Field for Format 0
18.	Not used
	Access Control
46.	Built by BACC or BACCL call

BFMT1
(Build a Format 1 Destination Descriptor)

3.8.7 BFMT1 — Build a Format 1 Destination Descriptor

Use:

Call BFMT1 from the source task to build a format 1 destination descriptor for the outgoing connect block. Use a format 1 descriptor to connect only to a target task that requires specification of a task name.

Formats:

FORTRAN: CALL BFMT1 ([*status*],*tgtblk*,*ndsz*,*ndname*,
objtype,*namesz*,*name*)

COBOL: CALL "BFMT1" USING [*status*],*tgtblk*,*ndsz*,*ndname*,
objtype,*namesz*,*name*.

BASIC: CALL BFMT1 BY REF ([*status*%],*tgtblk*$,*ndsz*%,*ndname*$,
objtype%,*namesz*%,*name*$)

Arguments:

* *status*

  specifies an integer variable containing completion status information on return from BFMT1. On return, the variable is set to .TRUE. (for FORTRAN) or to −1 (for COBOL and BASIC) if the BFMT1 call completed successfully. It is set to .FALSE. (for FORTRAN) or to 0 (for COBOL and BASIC) if there was an invalid BFMT1 argument.

  In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

* *tgtblk*

  specifies an array/string in which to build the destination descriptor. See the definition for your language in Section 3.8.1.

ndsz

specifies the node name length in bytes/characters. This field must be an integer variable or constant.

ndname

specifies the 1- to 6-element array/string that contains the name of the target node.

objtype

specifies the object type of the target task. For BFMT1, *objtype* must be 0.

namesz

specifies the length of the program name in bytes/characters. This field must be an integer variable or constant.

name

specifies a 1- to 6-element array/string containing the name of the target program.

Connect Block Offsets:

Length in Decimal Bytes/Characters	Destination Descriptor
6.	Destination node name with trailing blanks
1.	Descriptor format type (1 for BFMT1)
1.	Descriptor object type (0 for BFMT1)
	Descriptor Fields for Format 1
2.	Destination program name length (16. bytes/characters or less)
16.	Destination program name
	Access Control
46.	Built by BACC or BACCL call

Examples:

The following language-specific examples show the code for a BFMT1 call, including the declaration statements.

FORTRAN Example:

```
INTEGER*2 IOST(2),NDSIZ,OBJTY,PRSIZ
BYTE NDNAM(6),PRGNAM(5)
BYTE CONBLK(120)

DATA NDNAM/'T','A','C','O','M','A'/
DATA PRGNAM/'R','E','C','V','R'/

OBJTY=0
NDSIZ=6
PRSIZ=5
     .
     .
     .
CALL BFMT1 (IOST,CONBLK,NDSIZ,NDNAM,OBJTY,PRSIZ,PRGNAM)
```

COBOL Example:

```
WORKING-STORAGE SECTION.
                .
                .
                .
01 STORE-STUFF.
                .
                .
                .
   03 NODNAM   PIC X(6) VALUE "TACOMA".
   03 TSKNAM   PIC X(6) VALUE "RECVR".
   03 STAT     PIC X999 USAGE COMP.
   03 CONBLK   PIC X(120).
   03 NLENG    PIC 9 USAGE COMP.
   03 TLENG    PIC 9 USAGE COMP.
   03 DUMMY    PIC X (2).
                .
                .
                .
PROCEDURE DIVISION
                .
                .
                .
**********************************
* BUILD A FORMAT 1 CONNECT BLOCK. *
**********************************
     MOVE 6 TO NLENG.
     MOVE 5 TO TLENG.
     CALL BFMT1 USING
                 STAT
                 CONBLK
                 NLENG
                 NODNAM
                 DUMMY
                 TLENG
                 TSKNAM.
```

BASIC-PLUS-2 Example:

```
40 CONBLK$=STRING(120%,0%)

\ NDNAM.LEN%=6%
\ TSKNAM.LEN%=5%
\ NDNAM$="TACOMA"
\ TSKNAM$="RECVR"

\ CALL BFMT1 BY REF (STAT%,CONBLK$,NDNAM.LEN%,
                     NDNAM$,DUMMY%,TSKNAM.LEN%,
                     TSKNAM$)
```

CLSNT
(End Task Network Operations)

3.8.8 CLSNT — End Task Network Operations

Use:

Call CLSNT from either task to end that task's network activity, abort all of its logical links, and free all its network LUNs. If the CLSNT call occurs when data remains in the task's network data queue, network software:

- Reschedules the task if pending connect requests arrived while the task was active. The task receives these connect requests when it restarts. There is a limit of one retry.
- Rejects connect requests that arrived while the task was inactive.
- Discards interrupt, user disconnect, user abort, or network abort messages.

Formats:

FORTRAN: CALL CLSNT[W] [(*status*)]

COBOL: CALL "CLSNT[W]" USING [*status*].

BASIC: CALL CLSNT[W] BY REF [(*status*%())]

Arguments:

* *status*

specifies completion status information on return from CLSNT. See the definition for your language in Section 3.8.1.

Error/Completion Codes:

1	The call completed successfully.
−9	The task is not a network task; OPNNT did not execute successfully.
−10	The network is not accessed on this LUN.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

CONNT
(Request Logical Link Connection)

3.8.9 CONNT — Request Logical Link Connection

Use:

Call CONNT from the source task to request a logical link with the target task. Before calling CONNT, you must build a connect block. To build the connect block's access control area, you call BACC or BACCL; to build its destination descriptor, you call BFMT0 or BFMT1.

When a remote system receives a connect request, it checks the remote task. If the task is currently installed and inactive, the system automatically loads and activates it before passing the connect block to the task. After retrieving the connect request with a GNDNT call, the task either accepts (ACCNT) or rejects (REJNT) the request. You can send 1 to 16. bytes/characters of user data to and/or receive 1 to 16. bytes/characters of user data from the remote task when it accepts/rejects your connect request

Formats:

FORTRAN: CALL CONNT[W](*lun*,[*status*],*tgtblk*,[*outsize*,*outmessage*], [*insize*,*inmessage*])

COBOL: CALL "CONNT[W]" USING *lun*,[*status*],*tgtblk*, [*outsize*,*outmessage*], [*insize*,*inmessage*].

BASIC: CALL CONNT[W] BY REF (*lun*%,[*status*%( )],*tgtblk*$, [*outsize*%,*outmessage*$], [*insize*%,*inmessage*$])

Arguments:

lun

assigns the logical link number. This value must be an integer variable or constant. Use this LUN when referring to this logical link in any subsequent RECNT, SNDNT, XMINT, ABTNT, or DSCNT call.

* *status*

specifies an array/data item containing completion status information on return from CONNT. In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments. If specified, this 2-element integer array/data item contains the following values when the call completes:

- The first status word — *status*%(0) (BASIC) or *status*(1) (FORTRAN, COBOL) — contains an error/completion code, as shown in the list that follows.

- The contents of the second status word—*status*%(1) (BASIC) or *status*(2) (FORTRAN, COBOL) — depend on the error/completion code in the first status word, as shown in the list that follows.

These are the error/completion codes you can receive in the first status word and the corresponding contents of byte 0 in the second status word. Byte 1 of the second status word is always 0.

Error/Completion Code First Status Word	Contents of Byte 0 Second Status Word
Connection accepted	Received byte count
Connection accepted with data overrun	Received byte count
Connection rejected by user with data overrun	Received byte count
Connection rejected by DECnet	Reason for rejection (see Appendix A)
Connection rejected by user	Received byte count
Directive error	Directive error code
All other cases	0

tgtblk

specifies an array/string containing the explicit access control information area and destination descriptor. See the definition for your language in Section 3.8.1.

outsize,outmessage

define optional user data to send. See the definition in Section 3.8.1 but note the exception.

insize,inmessage

define user data you can receive from the target task. These are paired optional arguments; use both or omit both.

insize	specifies the length in bytes/characters of the user data to receive. It must be an integer variable or constant.
* *inmessage*	specifies the array/string that stores the user data sent by the target task. This is a 1- to 16.-element byte array for FORTRAN or a 1- to 16.-element character string for COBOL or BASIC.

Error/Completion Codes:

1	The call completed successfully.
2	The call completed successfully; the connection was accepted, but some returned optional data sent to the target task when you called CONNT was lost.
−1	System resources needed for the logical link are unavailable.
−4	The connection was rejected and some optional data was lost (the data sent to the target task when you called CONNT).
−5	Either an optional user data buffer exceeds 16. bytes/characters, or the field length count in the connect block is too large.
−7	The connection was rejected by the network (see the reject reason codes in Appendix A).
−8	A logical link has already been established on the specified LUN.
−9	The task is not a network task; OPNNT did not execute successfully.
−12	The connection was rejected by the remote user task.

−13	You are using an invalid buffer; the *tgtblk*, *inmessage*, or *outmessage* buffer is outside the user task address space or (for FORTRAN) *tgtblk* is not word aligned.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

DSCNT
(Disconnect a Logical Link)

3.8.10 DSCNT — Disconnect a Logical Link

Use:

Call DSCNT from either task to disconnect the logical link and free the logical unit number. This call lets all pending transmits complete. While they are completing, the task continues to receive messages. When the last transmit has completed, the task aborts all pending receives and disconnects the link. The I/O status block gives an abort status for each aborted receive.

When you call DSCNT, you can send 1 to 16. bytes/characters of user data to the target task (see the *outsize,outmessage* arguments).

Formats:

FORTRAN: CALL DSCNT[W] (*lun*,[*status*][,*outsize*,*outmessage*])

COBOL: CALL "DSCNT[W]" USING *lun*,[*status*][,*outsize*,*outmessage*].

BASIC: CALL DSCNT[W] BY REF (*lun*%,[*status*%( )],
[*outsize*%,*outmessage*$])

Arguments:

lun

specifies the logical link to disconnect. It must be an integer variable or constant. If you initiated the connection, enter the LUN you used in the CONNT call. If you accepted the connection, enter the LUN you used in the ACCNT call.

* *status*

specifies completion status information on return from DSCNT. See the definition for your language in Section 3.8.1.

outsize,outmessage

define optional user data to send. See the definition in Section 3.8.1.

Error/Completion Codes:

1	The call completed successfully.
−2	No logical link has been established on the specified LUN.
−5	The optional user data exceeds 16. bytes/characters.
−9	The task is not a network task; OPNNT did not execute successfully.
−10	The network is not accessed on this LUN.
−13	You are using an invalid buffer; the *outmessage* buffer is outside the user task address space.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

GLNNT
(Get Local Node Information)

3.8.11 GLNNT — Get Local Node Information

Use:

Issue GLN$ from either task to place the name and default NSP segment size of the local node in a specified buffer.

Getting the local node name can be helpful if two tasks on the same node use the network interface to communicate. Each task can issue GLN$ and use the returned local node name as the destination in a connect request. You can also use GLN$ in a task that displays the local node name.

The default NSP segment size tells you how NSP segments data transmitted on a logical link. By knowing the default NSP segment size, you can adjust the length of message blocks to transmit for most efficient use of transmit buffers (large data buffers).

Formats:

FORTRAN: CALL GLNNT[W] ([*status*],*buflen*,*buf*)

COBOL: CALL "GLNNT[W]" USING [*status*],*buflen*,*buf*.

BASIC: CALL GLNNT[W] BY REF ([*status*%( )],*buflen*%,*buf*$)

Arguments:

* *status*

specifies completion status information on return from GLNNT. See the definition for your language in Section 3.8.1.

buflen

specifies an array/string containing the received data length. If you specify 6 bytes/characters, only the local node name is returned. If you specify 8. bytes/characters, both the node name and the default NSP segment size are returned. This value must be an integer variable or constant.

* *buf*

specifies the array/string containing the received data. In FORTRAN, the buffer must start on an even byte (word) boundary. On return from the call, the data is stored as follows:

Length in Bytes/ Characters	Contents/Meaning
6	Local node name in ASCII (left justified and filled with spaces if the name is less than 6 bytes/characters)
2	Default NSP segment size

Error/Completion Codes:

1 The call completed successfully.

-4 Data overrun. The network data was longer than the specified buffer. As much data as fits into the buffer is transferred to it; any remaining data is lost.

-9 The task is not a network task; OPNNT did not execute successfully.

-10 The network is not accessed on this LUN.

-13 You are using an invalid buffer; the buffer specified to receive network data is outside the user task address space, or (for FORTRAN) it is not word aligned.

-40 A directive error has occurred. Directive error codes are defined in the *RSX-11M/M-PLUS Executive Reference Manual*.

GNDNT
(Get Network Data)

3.8.12 GNDNT — Get Network Data

Use:

Call GNDNT from either task to get data from that task's network data queue and store it in the specified mail buffer (see *mailbuf*). On completion, the variable specified by the *type* argument contains a code that indicates what type of message GNDNT retrieved. The code indicates one of the following unsolicited message types:

Connect request	*type* code 1
Interrupt message	*type* code 2
User disconnect notice	*type* code 3
User abort notice	*type* code 4
Network abort notice	*type* code 5

Only one GNDNT request can be outstanding. If you issue a GNDNT while another GNDNT is outstanding, your request completes with error code −14.

If GNDNT retrieves a connect request, it writes the accompanying connect block information to the mail buffer. You can use a long or short connect block depending on the length of the user IDs, passwords, and accounts you expect to receive. For information about the incoming connect block, see the "Connect Block" section of this call description.

Formats:

FORTRAN: CALL GNDNT[W] ([*status*],*type*,[*mailsz*],[*mailbuf*],
[*ltonly*],[*immed*][,*typmsk*])

COBOL: CALL "GNDNT[W]" USING [*status*],*type*,*mailsz*,*mailbuf*,
[*ltonly*],[*immed*][,*typmsk*].

BASIC: CALL GNDNT[W] BY REF ([*status*%( )],*type*%,*mailsz*%,
mailbuf$,[*ltonly*%],[*immed*%]
[,*typmsk*%])

Arguments:

* *status*

specifies a 2-element integer array/data item that contains completion status information on return from GNDNT. In COBOL and BASIC, you cannot omit *status*, but you can set its value to 0 to prevent the return of status information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments. If specified, this 2-element integer array/data item contains the following values when the call completes:

- The first status word — *status*%(0) (BASIC) or *status*(1) (FORTRAN, COBOL) — contains an error/completion code, as shown in Status Table A.

- The contents of the second status word — *status*%(1) (BASIC) or *status*(2) (FORTRAN, COBOL) — depend on the error/completion code in the first status word, as shown in Status Table A.

Status Table A shows the error/completion codes you can receive in the first status word and the corresponding contents of byte 0 in the second status word. Byte 1 of the second status word is always 0.

Status Table A

	First Status Word	Second Status Word
GNDNT completes with an error	–40 -*n* (other than –40)	Directive error code 0
GNDNT completes successfully and the *ltonly* flag is –1 (.TRUE.):	+*n*	The low-order byte contains the number of bytes/characters in the first network data item in the queue.
GNDNT completes successfully and the *ltonly* flag is 0 (.FALSE.):	+*n*	Depends on the data message type. For each message type, Status Table B lists the contents of the second status word.

Status Table B shows the contents of the second status word for each type of message successfully retrieved by GNDNT.

Status Table B

Type Code	Message Type	Low-Order Byte	High-Order Byte
1	Connect request	Number of bytes/characters in the connect block.	Access verification* and privilege code: 1 = Nonprivileged requesting user. 2 = Privileged requesting user. 0 = No verification done.* −1 = Verification failed.***
2	Interrupt message	Number of bytes/characters in the message. Zero indicates that no message was received.	LUN on which the notice was received.
3	User disconnect		
4	User disconnect		
5	Network abort	Reason for network abort. Refer to Appendix A for information on the codes.	LUN on which the notice was received.

* If access verification is enabled, the Network Verification Program evaluates the access control information in the connect request before passing the incoming request to the task's network data queue.

** The verification task was not installed on the target node, or it was set to OFF with the NCP SET EXECUTOR VERIFICATION command, or the proper access control file was not available.

*** Either the account is not in the system account file or the password does not match the one in the file.

* *type*

specifies an integer variable indicating a data message type code on return from GNDNT. The code indicates the type of data message GNDNT placed in the mail buffer. Status Table B lists the codes and tells what message types they represent.

mailsz

specifies the size of the task's mail buffer in bytes/characters. In FORTRAN, you can omit this integer variable or constant if you specify *ltonly* as .TRUE. In COBOL and BASIC, you can set it to 0 if you specify *ltonly* as –1. Otherwise, *mailsz* must be a value greater than 0.

The incoming data is written to the buffer according to the offsets appropriate to the connect block type that you specified in the OPNNT call.

You can allocate a mail buffer that is equal to, smaller than, or larger than the expected connect block and optional data. To receive an entire connect block, allocate space according to the N.CBL or M.CBL length:

Short connect block	98. bytes (N.CBL)
Long connect block	178. bytes (M.CBL)

You can add space for optional data:

Optional data	Up to 16. bytes
Optional data leangth field	2. bytes

Network software writes the retrieved information to the buffer field by field, according to the offsets of the specified connect block type. If the mail buffer and the incoming connect block are different sizes, the following results occur.

Mail Buffer Size	Result
You allocate a buffer that is large than the incoming connect block.	No error occurs.
You allocate a buffer that is smaller than a full connect block	Connect block data is written field by field into the buffer until no more fits. A data (IS.DAO) completion status results.

Mail Buffer Size	Result
You allocate a buffer for receiving a short connect block and instead receive a long connect block.	If the incoming data fits according to the short connect block offsets, you get all the data, but a data overrun (IS.DAO) completion status results. If the data in any incoming field exceeds the size of the analogous receiving field, the data in that field is lost. The length value for the field becomes 0, and a data overrun (IS.DAO) completion status results.

* *mailbuf*

specifies a 1- to *n*-element array/string containing the network data on return from GNDNT (see Table 3–2). In FORTRAN, this array must start on an even byte (word) boundary; you can omit it if you specify *ltonly* as .TRUE. In COBOL and BASIC, you can set it to 0 if you specify *ltonly* as –1.

ltonly

specifies dynamic assignment of mail buffer space. When you specify *ltonly* as .TRUE. (FORTRAN) or –1 (COBOL and BASIC), the *type* variable returns the type code of the first message in the network data queue, and the low-order byte of the second status word returns its length. The message is not removed from the queue or placed in the mail buffer.

If you specify the *typmsk* argument, you must specify *ltonly* as 0 in COBOL and BASIC; in FORTRAN, you must omit *ltonly* or specify it as .FALSE. In COBOL and BASIC, you can omit *ltonly* only if you omit all trailing arguments. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets in Section 3.7.

immed

specifies GNDNT action based on data in the network data queue.

Value of *immed*	Data in Network Queue?	GNDNT Action
.TRUE. (FORTRAN) or –1 (COBOL and BASIC)	Yes	Completes normally
	No	Completes with error code –6 (no data in queue)
.FALSE. (FORTRAN) or 0 (COBOL and BASIC) or omitted	Yes	Completes normally
	No	Does not complete until there is data in the queue

You cannot omit *immed* in COBOL or BASIC unless you also omit all trailing arguments. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets in Section 3.7.

typmsk

specifies the data type to select from the network data queue. Normally, GNDNT returns items from the network data queue on a first-in, first-out basis. However, *typmsk* lets you select the first item on the queue that matches a specific message type and/or LUN. You specify an integer variable or constant, as follows:

Message Type (Byte 0)		Logical Unit Number (Byte 1)
1	Connect request	0 (Selects the first LUN to request a connection.)
2	Interrupt message	0 or LUN
3	User disconnect	0 or LUN
4	User abort	0 or LUN
5	Network abort	0 or LUN
0	Selects any message type on the specified LUN	LUN

For example, to select the first interrupt message (type 2) on LUN 3 from the network data queue, you use a variable for the *typmsk* argument, declare it as an integer, and assign it a value. You code the argument as (3*256.) + 2.

Specifying 0 in byte 1 returns the first message of the type specified in byte 0, regardless of the LUN.

With *typmsk*, you must also include *mailsz* and *mailbuf*. In COBOL and BASIC, you must also specify *ltonly* as 0. In FORTRAN, you can omit *ltonly* or specify it as .FALSE.

Connect Block:

Table 3–2 lists the contents of an incoming connect block. The access control fields differ according to the connect block size that you specified in the OPNNT call.

The source descriptor differs according to the source system type. If the source is an RSX system

- and the source node did not send proxy information, you receive a Format 1 source descriptor containing the ASCII source task name.
- and the source node sent proxy information, you receive a Format 2 source descriptor containing the proxy information.

Table 3–2: Incoming Connect Block

Length in Decimal Bytes/ Characters	Contents
2.	Temporary logical link address (required by the network; do not modify)
2.	NSP segment size (used by NSP to send message data to source)
	DESTINATION DESCRIPTOR (20.-byte/character total)
1.	Destination descriptor format type 0 for BFMT0, or 1 for BFMT1)
	Descriptor Field for Format 0
18.	Not used

(continued on next page)

Table 3–2 (Cont.): Incoming Connect Block

Length in Decimal Bytes/ Characters	Contents
	Descriptor Fields for Format 1
2.	Destination program name length (equal to or less than 16. bytes/characters)
16.	Destination program name
	SOURCE DESCRIPTOR (26.-byte/character total)
6.	Source node name (name of node requesting the connection; ASCII, with trailing blanks
1.	Source descriptor format type (format 0, 1, or 2)
1.	Source object type (object type of program requesting connection: 1–255 for format 0, or 0 for format 1 or 2)
	Descriptor Field for Format 0
18.	Not used
	Descriptor Fields for Format 1
2.	Source descriptor length (equal to or less than 16. bytes/characters)
16.	Source descriptor
	Descriptor Fields for Format 2
2.	Binary UIC group identifier
2.	Binary UIC member identifier
2.	Source descriptor length (12. bytes or less)
12.	Source descriptor

Table 3–2 (Cont.): Incoming Connect Block

Length in Decimal Bytes/ Characters	Contents
	ACCESS CONTROL (46.-byte/character total)
	If no verification performed
2.	Source program user ID length (equal to or less than 16. bytes/characters for a short connect block or 40. characters for a long connect block)
16.or	Source program user ID, short connect block
39. +	Source program user ID, long connect block
1.	Not used
2.	Source program password length (16. bytes/characters or less for a short connect block or 40. characters or less for a long connect block)
8. or	Source program password, short connect block
39. +	Source program password, long connect block
1.	Not used
2.	Account number length (16. bytes/characters or less for a short connect block or 40. characters or less for a long connect block)
16. or	Account number, short connect block
39. +	Account number, long connect block
1.	Not used
	If verification performed
2.	Default device name for destination program
1.	Default device unit number
1.	Not used
2.	Log-in UIC from account file used for destination program
11.	Default directory string (0 if no default string)
29.	Not used

(continued on next page)

Table 3–2 (Cont.): Incoming Connect Block

Length in Decimal Bytes/ Characters	Contents
	OPTIONAL DATA (18.-byte/character total)
2.	Length of optional user data (16. bytes/characters or less; 0 if no optional data)
16.	Optional user data sent by source program (0 to 16. bytes/characters)

Error/Completion Codes:

1 The call completed successfully.

2 The call completed successfully, but some returned optional data was lost.

−4 Data overrun. The network data was longer than the mail buffer. As much data as fits into the mail buffer is transferred to it; any remaining data is lost.

−6 There is no data in the network data queue to return.

−9 The task is not a network task; either OPNNT did not execute successfully, or CLSNT was issued with this GNDNT pending.

−10 The network is not accessed on this LUN.

−13 You are using an invalid buffer; the mail buffer is outside the user task address space, or (for FORTRAN) it is not word aligned.

−14 A GNDNT is already pending.

−40 A directive error has occurred. Directive error codes are defined in the *RSX–11M/M–PLUS Executive Reference Manual.*

OPNNT
(Access the Network)

3.8.13 OPNNT — Access the Network

Use:

Call OPNNT to establish the task as an active network task and create the task's network data queue. Call OPNNT before any other network subroutine.

Formats:

FORTRAN: CALL OPNNT[W] ([*lun*],[*status*],[*mstat*],[*count*],[*lrp*][,mbxflg])

COBOL: CALL "OPNNT[W]" USING [*lun*],[*status*],[*mstat*], [*count*],[*lrp*][,mbxflg].

BASIC: CALL OPNNT[W] BY REF ([*lun*%],[*status*%( )],[*mstat*%( )], [*count*],[*lrp*%][,mbxflg])

Arguments:

lun

specifies a logical unit number for the task's network data queue. This value must be an integer variable or constant. You can omit this argument if you have already assigned the LUN to NS: by using the GBLDEF option of .MBXLU at task build time (Section 1.2.1). When you omit *lun* in COBOL or BASIC, you must also omit all trailing arguments. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets in Section 3.7.

* *status*

specifies completion status information on return from OPNNT. See the definition for your language in Section 3.8.1.

* *mstat*

specifies a 3-element integer array (or elementary numeric data item for COBOL) to contain current status information for the task's network data queue. When specified (+), the *mstat* array/data item is updated whenever data arrives or is retrieved by GNDNT. Do not use this array/data item for other purposes while the task is active on the network.

Values returned in this array/data item are:

mstat(1) Number of items in the network data queue

mstat(2) Data type of the first data item:

1 – Connect request
2 – Interrupt message
3 – User disconnect
4 – User abort
5 – Network abort

mstat(3) Length of first data item

count

specifies the maximum number of simultaneously active connections the task accepts. When the number of active logical links equals the *count* value, the network rejects any incoming connect request. This integer variable or constant must not exceed 255 (decimal). A value of 0 (which is also the default) sets no limit as long as network resources are available.

To prevent access to your task, specify a *count* value of 1 so that GNDNT rejects all incoming connect requests. You can still establish outgoing links by using CONNT.

lrp

specifies the link recovery period. The link recovery period is the number of minutes that elapses from the time of a physical link failure until the associated logical link is irrecoverable. This integer variable or constant must be in the range of 0 through 32767 (decimal).

When specifying an *lrp* value, remember that your task is locked in memory until the link recovery period has elapsed if the task has outstanding I/O when the link fails. This can cause serious delays for other system users who need to access the occupied area of memory.

mbxflg

specifies that the task has a mail buffer that supports sending and receiving long connect blocks. For a task that uses long connect blocks, set the *mbxflg* value to an integer variable or constant with the value 1. For a task that uses short connect blocks, omit the argument.

Error/Completion Codes:

1	The call completed successfully.
−1	System resources needed for the network data queue are not available.
−10	The network is being dismounted, or the user task has already accessed the network.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

RECNT
(Receive Data over a Logical Link)

3.8.14 RECNT — Receive Data over a Logical Link

Use:

Call RECNT from either task to receive data over an established logical link and store it in a specified buffer.

Formats:

FORTRAN: CALL RECNT[W] (*lun*,[*status*],*insize*,*indata*)

COBOL: CALL "RECNT[W]" USING *lun*,[*status*],*insize*,*indata*.

BASIC: CALL RECNT[W] BY REF (*lun*%,[*status*%( )],*insize*%,*indata*$)

Arguments:

lun

specifies the logical unit number for the logical link over which to receive data. It must be an integer variable or constant. If you initiated the connection, enter the LUN you used in the CONNT call. If you accepted the connection, enter the LUN you used in the ACCNT call.

* *status*

specifies completion status information on return from RECNT. See the definition for your language in Section 3.8.1 but note this addition: If a positive value or −4 (data overrun) is returned in the first status word, the second status word contains the number of bytes/characters of data received.

insize

specifies the receive data buffer length in bytes/characters. This integer variable or constant can be a maximum of 8128 (decimal).

* *indata*

specifies the array/string containing the received message data.

Error/Completion Codes:

1	The call completed successfully.
-2	No logical link has been established on the specified LUN.
-3	The logical link was disconnected during I/O operations.
-4	Data overrun. More message data was transmitted than requested. As much data as fits into the receive buffer is transferred to it; any remaining data is lost.
-9	The task is not a network task; OPNNT did not execute successfully.
-13	You are using an invalid buffer; either the *indata* array/string is outside the user task address space, or the buffer size (*insize*) exceeds 8128. bytes/characters.
-40	A directive error has occurred. Directive error codes are defined in the *RSX-11M/M-PLUS Executive Reference Manual*.

REJNT
(Reject Logical Link Connect Request)

3.8.15 REJNT — Reject Logical Link Connect Request

Use:

Call REJNT from the target task to reject a logical link connect request. With REJNT, you can send 1 to 16. bytes/characters of user data to the requesting task (see the *outsize,outmessage* arguments).

Formats:

FORTRAN: CALL REJNT[W] ([*status*],*mailbuf*[,*outsize,outmessage*])

COBOL: CALL "REJNT[W]" USING [*status*],*mailbuf*,[*outsize,outmessage*].

BASIC: CALL REJNT[W] BY REF ([*status*%( )],*mailbuf*$,
[*outsize*%,*outmessage*$])

Arguments:

* *status*

specifies completion status information on return from REJNT. See the definition for your language in Section 3.8.1.

mailbuf

specifies the 1- to *n*-element array/string containing information necessary to reject the connect request. In FORTRAN, this array must start on an even byte (word) boundary. GNDNT refers to this same array/string.

outsize,outmessage

define optional user data to send. See the definition in Section 3.8.1.

Error/Completion Codes:

1	The call completed successfully.
−3	The task that requested the connection has aborted or has requested a disconnect before the connection could complete.
−5	Either the temporary link address in the mail buffer is not valid, or the optional user data buffer exceeds 16. bytes/characters.
−9	The task is not a network task; OPNNT did not execute successfully.
−10	The network is not accessed on this LUN.
−13	You are using an invalid buffer; the *mailbuf* or *outmessage* array/string is outside the user task address space, or (for FORTRAN) the *mailbuf* array is not word aligned.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

SNDNT
(Send Data over a Logical Link)

3.8.16 SNDNT — Send Data over a Logical Link

Use:

Call SNDNT from either task to send message data over the logical link.

Formats:

FORTRAN: CALL SNDNT[W] (*lun*,[*status*],*outsize*,*outdata*)

COBOL: CALL "SNDNT[W]" USING *lun*,[*status*],*outsize*,*outdata*.

BASIC: CALL SNDNT[W] BY REF (*lun*%,[*status*%( )],*outsize*%,*outdata*$)

Arguments:

lun

specifies the logical unit number for the logical link over which to send data. This value must be an integer variable or constant. If you initiated the connection, enter the LUN you used in the CONNT call. If you accepted the connection, enter the LUN you used in the ACNNT call.

* *status*

specifies completion status information on return from SNDNT. See the definition for your language in Section 3.8.1 but note this addition: If 1 is returned in the first status word, the second status word contains the number of bytes/characters of transmitted data.

outsize

specifies the length in bytes/characters of the data to send. This integer variable or constant can be a maximum of 8128 (decimal).

outdata

specifies a 1- to *n*-element array/string containing the message data to send.

Error/Completion Codes:

1	The call completed successfully.
−2	No logical link has been established on the specified LUN.
−3	The logical link was disconnected during I/O operations.
−9	The task is not a network task; OPNNT did not execute successfully.
−13	You are using an invalid buffer; either the *outdata* array/string is outside the user task address space, or the buffer size (*outsize*) exceeds 8128. bytes/characters.
−40	A directive error has occurred. Directive error codes are defined in the *RSX−11M/M−PLUS Executive Reference Manual*.

WAITNT
(Suspend the Calling Task)

3.8.17 WAITNT — Suspend the Calling Task

Use:

Call WAITNT from any task to suspend that task's operation until completion of a call specified by one of the associated status blocks.

Formats:

FORTRAN: CALL WAITNT ([*index*],*status1*,...,*statusn*)

COBOL: CALL "WAITNT" USING [*index*],*status1*,...,*statusn*.

BASIC: CALL WAITNT BY REF ([*index*%],*status1*%( ),...,*statusn*%( ))

Arguments:

* *index*

specifies an integer variable containing the positional number of the status block associated with the call that has completed.

In COBOL and BASIC, you cannot omit *index*, but you can specify 0 for *index* to prevent the return of index information. See the discussion of square brackets in Section 3.7 for more information on omitting optional arguments.

status1,...,*statusn*

specify one or more status blocks. WAITNT completes when any one of the calls associated with a status block in this list completes.

XMINT
(Send Interrupt Message)

3.8.18 XMINT — Send Interrupt Message

Use:

Call XMINT from either task to send an interrupt message over an established logical link. This call places the message you send on the target task's network data queue. The target task must issue a GNDNT call to retrieve the message before you can issue another XMINT.

Formats:

FORTRAN: CALL XMINT[W] (*lun*,[*status*],*intsize*,*intmsg*)

COBOL: CALL "XMINT[W]" USING *lun*,[*status*],*intsize*,*intmsg*.

BASIC: CALL XMINT[W] BY REF (*lun*%,[*status*%( )],*intsize*%,*intmsg*$)

Arguments:

lun

specifies the logical unit number for the logical link over which to send the interrupt message. This value must be an integer variable or constant. If you initiated the connection, enter the LUN you used in the CONNT call. If you accepted the connection, enter the LUN you used in the ACCNT call.

* *status*

specifies completion status information on return from XMINT. See the definition for your language in Section 3.8.1.

intsize

specifies the length in bytes/characters of the interrupt message to send. It must be an integer variable or constant.

intmsg

specifies a 1- to 16.-element array/string containing the interrupt message to send.

Error/Completion Codes:

1	The call completed successfully.
-2	No logical link has been established on the specified LUN.
-3	The logical link was disconnected during I/O operations.
-5	The interrupt message exceeds 16. bytes/characters.
-9	The task is not a network task; OPNNT did not execute successfully.
-11	An interrupt message was transmitted before a previous interrupt message had been received by the remote task.
-13	You are using an invalid array/string; the *intmsg* array/string is outside the user task address space.
-40	A directive error has occurred. Directive error codes are defined in the *RSX-11M/M-PLUS Executive Reference Manual*.

3.8.19 FORTRAN Intertask Communication Programming Examples

The following two programs are examples of FORTRAN intertask communication. They are cooperating tasks. FTNTRN is a transmit task; FTNREC is a receiver task.

These programming examples are included in your tape or disk kit.

3.8.19.1 Transmit Example

The FTNTRN program accesses the network, connects to FTNREC, transmits inquiries to FTNREC, and processes responses. When FTNTRN completes sending inquiries, it disconnects the link, stops accessing the network, and exits.

```
C
C
C Copyright (C) 1983, 1985, 1986, 1987 by
C Digital Equipment Corporation, Maynard, Mass.
C
C
C This software is furnished under a license and may be used and  copied
C only  in  accordance  with  the  terms  of  such  license and with the
C inclusion of the above copyright notice.  This software or  any  other
C copies  thereof may not be provided or otherwise made available to any
C other person.  No title to and ownership of  the  software  is  hereby
C transferred.
C
C The information in this software is subject to change  without  notice
C and  should  not  be  construed  as  a commitment by Digital Equipment
C Corporation.
C
C Digital assumes no responsibility for the use or  reliability  of  its
C software on equipment which is not supplied by Digital.
C
C
C
C
C   FTNTRN - Transmit inquiries to FTNREC and process responses
C
C   To task build, use the following command string:
C
C   FTNTRN,FTNTRN = FTNTRN
C   LB:[1,1]NETFOR/LB
C   LB:[1,1]F4POTS/LB
C   LB:[1,1]RMSLIB/LB           (if RMS is included)
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                 (if RMS is included)
C   //
C
C   Note: This task uses a long connect block.
C
        INTEGER         MLTYP,RECSIZ,SNDSIZ,MESNUM,XMITS,NDLEN,TSKLEN
        INTEGER         MBXFLG,NETLUN,LNKLUN,IOST(2),MSTAT(3)
        BYTE            ERRMES(2),TSKNAM(6),NDNAM(6),DEFNOD(6),DEFTSK(6)
        BYTE            CONBLK(152),SNDBUF(50),RECBUF(10)
        LOGICAL*1       STAT,IMMED
C
C   Specify the default node and task names
C
        DATA            DEFNOD /'R','E','M','N','O','D'/
        DATA            DEFTSK /'R','E','C','V','E','R'/
C
C   Specify flags for long connect block and immediate mode
C
        DATA            MBXFLG,IMMED /1,.TRUE./
C
```

```
C   LUNs for the network mailbox and the logical link
C
        DATA            NETLUN,LNKLUN /1,2/
C
C   Specify transmit count, send buffer size, and receive buffer size
C
        DATA            XMITS,SNDSIZ,RECSIZ /20,50,10/
C
C  Get the node and task names
C
10      TYPE 250                        !* Ask for node-name
        READ(5,260) (NDNAM(NDLEN),NDLEN=1,6) !* Get the name
        DO 20 NDLEN=6,1,-1              !* Loop to find length of name
        IF (NDNAM(NDLEN).NE.' ') GOTO 40!* If not a space, get task-name
20      CONTINUE
        DO 30 I=1,6
30      NDNAM(I)=DEFNOD(I)              !* Default node name 'MASTER'
        NDLEN=6                         !* Length of default name

40      TYPE 270                        !* Ask for the task-name
        READ(5,260) (TSKNAM(TSKLEN),TSKLEN=1,6) !* Get it
        DO 50 TSKLEN=6,1,-1             !* TSKLEN is length of task-name
        IF (TSKNAM(TSKLEN).NE.' ') GOTO 70 !* If not space, access network
50      CONTINUE
        DO 60 I=1,6
60      TSKNAM(I)=DEFTSK(I)             !* Default task name 'RECVER'
        TSKLEN=6                        !* Length of default name
C
C  Access network - MBXFLG indicates that we will use a long connect block
C
70      CALL    OPNNTW(NETLUN,IOST,MSTAT,,,MBXFLG)
        IF      (IOST(1).NE.1)GOTO 140  !* If failure, just exit
C
C  Build a Format 1 connect block
C
        CALL    BFMT1(STAT,CONBLK,NDLEN,NDNAM,,TSKLEN,TSKNAM)
        IF      (STAT)GOTO 80           !* If success, go on
        TYPE    200                     !* Else, type out a failure
                                        !*   notification
        GOTO    130                     !*   and exit
C
C  Connect to the task on the remote node
C
80      CALL    CONNTW(LNKLUN,IOST,CONBLK)
        IF      (IOST(1).EQ.1)GOTO 90   !* If success, confirm it
        TYPE    240,IOST                !* Else print status block
        GOTO    130                     !* Deaccess the network
                                        !*   and exit
90      TYPE    220                     !* Print connect confirmation
                                        !*   to network and exit

C
C  Send and receive messages to and from the remote node
C
        DO      120 MESNUM=1,XMITS
```

(continued on next page)

```
C
C  First get any error messages sent from the other side in interrupt
C  messages
C
        IF      (MSTAT(1).EQ.0)GOTO 100 !* If MSTAT(1)=0 no messages
                                        !*   are there
        CALL    GNDNTW(IOST,MLTYP,2,ERRMES,,IMMED,2) !* Get the message
        IF      (IOST(1).NE.1)GOTO 100  !* If we couldn't get the message,
                                        !*   just ignore it
        TYPE    210,ERRMES(1)           !* Print out the message
C
C  Send the inquiry
C
100     CALL    SNDNTW(LNKLUN,IOST,SNDSIZ,SNDBUF)
        IF      (IOST(1).EQ.1)GOTO 110  !* If success, continue
        TYPE    210,MESNUM              !* Otherwise, type out an
                                        !*   error message
        GOTO    120                     !*   and start a new message
C
C  Receive the response from the remote node
C
110     CALL    RECNTW(LNKLUN,IOST,RECSIZ,RECBUF)
        IF      (IOST(1).EQ.1)GOTO 120  !* If success, continue
        TYPE    210,MESNUM              !* Otherwise, type out an
                                        !*   error message
120     CONTINUE
C
C  Disconnect the link
C
        TYPE    230                     !* Print out disconnect message
        CALL    DSCNTW(LNKLUN,IOST)
C
C  Come here to close the network and exit
C
130     CALL    CLSNTW
140     STOP    'End of program execution'
C
C  FORMAT Statements
C
200     FORMAT  (' Error building connect block')
210     FORMAT  (' Error on inquiry  ',I3)
220     FORMAT  (' Link enabled')
230     FORMAT  (' Link disabled')
240     FORMAT  (' Connect fail: IOST = ',2I6)
250     FORMAT  ('$Node-name <REMNOD>: ')
260     FORMAT  (6A1)
270     FORMAT  ('$Task-name <RECVER>: ')
        END
```

3.8.19.2 Receive Example

The FTNREC program receives inquiries from FTNTRN. It returns errors to FTNTRN as interrupt messages, which FTNTRN then displays on the terminal.

```
C
C
C
C Copyright (C) 1983, 1985, 1986, 1987 by
C Digital Equipment Corporation, Maynard, Mass.
C
C
C This software is furnished under a license and may be used and  copied
C only  in  accordance  with  the  terms  of  such  license and with the
C inclusion of the above copyright notice.  This software or  any  other
C copies  thereof may not be provided or otherwise made available to any
C other person.  No title to and ownership of  the  software  is  hereby
C transferred.
C
C The information in this software is subject to change  without  notice
C and  should  not  be  construed  as  a commitment by Digital Equipment
C Corporation.
C
C Digital assumes no responsibility for the use or  reliability  of  its
C software on equipment which is not supplied by Digital.
C
C
C   FTNREC.FTN - Receive inquiries from FTNTRN and send back responses
C
C   To task build, use the following command string:
C
C   FTNREC,FTNREC = FTNREC
C   LB:[1,1]F4POTS/LB
C   LB:[1,1]NETFOR/LB
C   LB:[1,1]RMSLIB/LB           (if RMS is included)
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                 (if RMS is included)
C   TASK=RECVER
C   //
C
C   Note: This task uses a long connect block.
C
      INTEGER           NETLUN,LNKLUN,MLTYP,INDEX,NUMBER,NUMMES
      INTEGER           MBXFLG,RECSIZ,SNDSIZ,INTSIZ
      INTEGER           MSTAT(3),IOST(2),IOST1(2),IOST2(2)
      BYTE              RECBUF(50),SNDDAT(10),MLBX(178),INTMES(2)
C
C   Specify LUNs for the network and the logical link
C
      DATA      NETLUN,LNKLUN /1,2/
C
C   Specify sizes for the receive buffer, send buffer, and interrupt buffer
C
      DATA      RECSIZ,SNDSIZ,INTSIZ /50,10,2/
C
C   Initialize some variables
C
```

(continued on next page)

```
      NUMMES = 0                                !* Number of messages received
      MBXFLG = 1                                !* Use long connect block
C
C  Access the network - use the long connect block
C
      CALL OPNNTW(NETLUN,IOST,MSTAT,,,MBXFLG)
      IF (IOST(1) .EQ. 1) GOTO 8
      TYPE *, 'Cannot access the network, status = ', ISTAT
      GOTO 100

8     IF (MSTAT(1) .EQ. 0) GOTO 40              !* If nothing in mailbox,
                                                !*   just close and exit
10    CALL GNDNT(IOST1,MLTYP,178,MLBX)  !* Issue Get Network Data
20    CALL WAITNT(INDEX,IOST1,IOST2)            !* Wait for a completion
      IF (INDEX .EQ. 2) GOTO 50         !* If INDEX=2, a receive
                                                !*   has been completed
C
C  We've received network data
C
      IF (I0ST1(1) .NE. 1) GOTO 40              !* If GNDNT failed, close
                                                !*   network and exit
      IF (MLTYP .GE. 3) GOTO 40         !* If MLTYP>=3 the link has
                                                !*   been broken
      IF (MLTYP .EQ. 2) GOTO 10         !* If MLTYP=2 we've received
                                                !*   an interrupt message
                                                !*   -- issue a new GNDNT
C
C  We've received a connect request - issue an accept
C
      CALL      ACCNTW(LNKLUN,IOST,MLBX)
      IF        (IOST(1) .NE. 1) GOTO 10        !* If failure, issue
                                                !*   a new GNDNT
C
C  Issue a receive to pick up data
C
30    CALL      RECNT(LNKLUN,IOST2,RECSIZ,RECBUF)
      GOTO       10                             !* Issue a new receive and
                                                !*   wait for a completion
C
C  Come here upon receiving a disconnect or abort
C
40    CALL      CLSNTW                          !* Close the network
      GOTO      100                             !*   and exit
C
C  Come here if we receive an inquiry
C
50    NUMMES=NUMMES+1                           !* Update the message count
      IF (IOST2(1) .EQ. 1) GOTO 60              !* If IOST2(1)-1 all's okay
C
C  If there was an error, return an interrupt message with message number
C
      INTMES(1)=NUMMES                  !* Send the message number
      CALL XMINT(LNKLUN,IOST,INTSIZ,INTMES)     !* Issue a new receive
      GOTO 70                                   !
C
```

```
C  Here the user can look at the data received in RECBUF and respond by
C  placing the requested information into SNDDAT
C
C  Send back the data and issue a new receive
C
60     CALL SNDNTW(LNKLUN,IOST,SNDSIZ,SNDDAT)
70     CALL RECNT(LNKLUN,IOST2,RECSIZ,RECBUF)
       GOTO 20                                      !* Wait for a completion
C
C  Exit program
C
100    STOP 'End of program execution'              !* Halt the program
       END                                          !*   and exit
```

3.8.20 COBOL Intertask Communication Programming Examples

The following two programs are examples of COBOL intertask communication. They are cooperating tasks. COBTRN is a transmit task; COBREC is a receiver task.

These programming examples are included in your tape or disk kit.

3.8.20.1 Transmit Example

This program sends inquiries to the cooperating COBREC program on a remote node.

```
*
* Copyright (C) 1983, 1985, 1986, 1987 by
* Digital Equipment Corporation, Maynard, Mass.
*
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by Digital Equipment
* Corporation.
*
* Digital assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by Digital.
*

IDENTIFICATION DIVISION.
PROGRAM-ID. COBTRN.

*****************************************************************
*                                                               *
*    This is the transmit program of the DECnet intertask       *
*    communication example programs for COBOL.                  *
*                                                               *
*    To task build, use the following command string:           *
*                                                               *
*   COBTRN,COBTRN=COBTRN,[1,1]NETFOR/LB,C81LIB/LB,RMSLIB/LB     *
*   /                                                           *
*   UNITS=10                                                    *
*   ACTFIL=4                                                    *
*   EXTTSK=1000                 (if RMS is included)            *
*   //                                                          *
*                                                               *
*    This task illustrates the use of a long connect block.     *
*                                                               *
*****************************************************************

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT DUMMY-FILE ASSIGN TO "COBTRN.DUM".

DATA DIVISION.
FILE SECTION.
FD      DUMMY-FILE
        LABEL RECORD STANDARD.
01      DUMMY-FILE-REC.
```

(continued on next page)

```
  02    FILLER          PIC X(132).

WORKING-STORAGE SECTION.
01  MSGS.
    03  MSG1.
        05  FILLER      PIC X(32)   VALUE " NETWORK OPEN FAILED,
-                                       "IOST(1) = ".
        05  MSG1-STAT1  PIC -99999.
        05  FILLER      PIC X(11)   VALUE " IOST(2) = ".
        05  MSG1-STAT2  PIC -99999.
    03  MSG2.
        05  FILLER      PIC X(25)   VALUE " CONNECT FAIL, IOST(1)
-                                       " = ".
        05  MSG2-STAT1  PIC -99999.
        05  FILLER      PIC X(11)   VALUE " IOST(2) = ".
        05  MSG2-STAT2  PIC -99999.
    03  MSG3.
        05  FILLER      PIC X(20)   VALUE " ERROR ON INQUIRY # ".
        05  MSG3-ERR1   PIC X(2).
    03  MSG4.
        05  FILLER      PIC X(31)   VALUE " ERROR ON INQUIRY DURI
-                                       "NG SEND: ".
        05  MSG4-NUM1   PIC 99.
    03  MSG5.
        05  FILLER      PIC X(34)   VALUE " ERROR ON INQUIRY DURI
-                                       "NG RECEIVE: ".
        05  MSG5-NUM1   PIC 99.
01  ARRAYS.
    03  IOST.
        05  IOSTAT OCCURS 2 TIMES PIC S9999 USAGE COMP.
    03  MSTAT.
        05  MSTATS OCCURS 3 TIMES PIC S9999 USAGE COMP.
01  STORE-STUFF.
    03  MBXFLG          PIC 99      COMP VALUE 1.
    03  TEN             PIC 99      COMP VALUE 10.
    03  OPNLUN          PIC 99      COMP VALUE 2.
    03  RESULT-REC      PIC X(80).
    03  IN-FILE         PIC X(6).
    03  NODNAM          PIC X(6).
    03  TSKNAM          PIC X(9).
    03  FILLER          PIC X.
    03  STAT            PIC S999    USAGE COMP.
    03  CONBLK          PIC X(72).
    03  NLENG           PIC 9       USAGE COMP.
    03  TLENG           PIC 9       USAGE COMP.
    03  CONLUN          PIC 99      COMP VALUE 3.
    03  XMITS           PIC 99      COMP VALUE 20.
    03  MESNUM          PIC 99.
    03  MLTYP           PIC 9.
    03  FILLER          PIC 9.
    03  MLBXSZ          PIC 99      COMP VALUE 2.
    03  ERRMES          PIC X(2).
    03  DUMMY           PIC X(2).
    03  IMMED           PIC S      COMP VALUE -1.
    03  TYPMSK          PIC S99999.
```

```
    03  FILLER          PIC 9.
    03  SNDSIZ          PIC 99      COMP VALUE 50.
    03  SNDBUF          PIC X(50).
    03  RECSIZ          PIC 99      COMP VALUE 10.
    03  RECBUF          PIC X(10).
PROCEDURE DIVISION.
A100-START.

*****************************************************************
*                                                               *
*       Input node name and receiver task name from             *
*       terminal.                                               *
*                                                               *
*****************************************************************

        DISPLAY "ENTER NODE-NAME <MASTER>".
        ACCEPT IN-FILE.
        MOVE IN-FILE TO NODNAM.
        DISPLAY "ENTER TASK-NAME <RECVER>".
        ACCEPT IN-FILE.
        MOVE IN-FILE TO TSKNAM.

*****************************************************************
*                                                               *
*       Access the network.  If the access is unsuccessful,     *
*       print an error message and exit.                        *
*                                                               *
*****************************************************************

        CALL "OPNNTW" USING
                OPNLUN
                IOST
                MSTAT
                TEN
                0
                MBXFLG.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG1-STAT1
                MOVE IOSTAT (2) TO MSG1-STAT2
                DISPLAY MSG1
                GO C000-END.

*****************************************************************
*                                                               *
*       Build a FORMAT 1 connect block.  If the call did        *
*       not complete successfully, print an error message       *
*       and deaccess the network.                               *
*                                                               *
*****************************************************************
        MOVE 6 TO NLENG.
        MOVE 6 TO TLENG.
        CALL "BFMT1" USING
                STAT
```

(continued on next page)

```
                CONBLK
                NLENG
                NODNAM
                DUMMY
                TLENG
                TSKNAM
        IF STAT NOT = 0
                NEXT SENTENCE
        ELSE
                DISPLAY "ERROR BUILDING CONNECT BLOCK"
                GO B100-CLOSE.

*****************************************************************
*                                                               *
*       Connect to the task on the remote node.  If the         *
*       call completes unsuccessfully, print an error message   *
*       and close the network.  Otherwise, print "Link          *
*       enabled" message.                                       *
*                                                               *
*****************************************************************

        CALL "CONNTW" USING
                CONLUN
                IOST
                CONBLK.
        IF IOSTAT(1) = 1
                NEXT SENTENCE
        ELSE
                MOVE SPACES TO RESULT-REC
                MOVE IOSTAT(1) TO MSG2-STAT1
                MOVE IOSTAT(2) TO MSG2-STAT2
                MOVE MSG2 TO RESULT-REC
                DISPLAY RESULT-REC
                GO B100-CLOSE.
        DISPLAY "LINK ENABLED".

*****************************************************************
*                                                               *
*       Send and receive messages from the remote node.         *
*       If there is something on the network data queue         *
*       (MSTATS (1) > 0), get the message.                      *
*                                                               *
*****************************************************************

        PERFORM LOOP VARYING MESNUM FROM 1 BY 1 UNTIL MESNUM = XMITS.
LOOP.
        IF MSTATS(1) = 0
                NEXT SENTENCE
        ELSE
                CALL "GNDNTW" USING
                        IOST
                        MLTYP
                        MLBXSZ
                        ERRMES
                        DUMMY
```

```
                IMMED
                TYPMSK
        IF IOSTAT(1) = 1
                NEXT SENTENCE
        ELSE
                MOVE SPACES TO RESULT-REC
                MOVE ERRMES TO MSG3-ERR1
                MOVE MSG3 TO RESULT-REC
                DISPLAY RESULT-REC.

*****************************************************************
*                                                               *
*       Send a message to the task on the remote node.  If      *
*       unsuccessful, print an error message and start the      *
*       next transmission.                                      *
*                                                               *
*****************************************************************

        CALL "SNDNTW" USING
             CONLUN
             IOST
             SNDSIZ
             SNDBUF.
        IF IOSTAT(1) = 1
                NEXT SENTENCE
        ELSE
                MOVE SPACES TO RESULT-REC
                MOVE MESNUM TO MSG4-NUM1
                MOVE MSG4 TO RESULT-REC
                DISPLAY RESULT-REC
                GO LOOP.

*****************************************************************
*                                                               *
*       Receive a message from the remote node.  If             *
*       unsuccessful, print an error message and start          *
*       the next transmission.  If successful, simply           *
*       start the next transmission.                            *
*                                                               *
*****************************************************************

        CALL "RECNTW" USING
                CONLUN
                IOST
                RECSIZ
                RECBUF.
        IF IOSTAT(1) = 1
                NEXT SENTENCE
        ELSE
                MOVE SPACES TO RESULT-REC
                MOVE MESNUM TO MSG5-NUM1
                MOVE MSG5 TO RESULT-REC
                DISPLAY RESULT-REC.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       Deaccess the network.                                   *
*                                                               *
*****************************************************************

B000-ENDLOOP.
        DISPLAY "LINK DISABLED".
        CALL "DSCNTW" USING
                CONLUN
                IOST.

*****************************************************************
*                                                               *
*       Close the network and exit.                             *
*                                                               *
*****************************************************************

B100-CLOSE.
        CALL "CLSNTW".
        DISPLAY "END OF EXECUTION".
C000-END.
        STOP RUN.
```

3.8.20.2 Receive Example

The COBREC program receives inquiries from COBTRN. It returns errors to COBTRN as interrupt messages, which COBTRN then displays on the terminal.

```
*
* Copyright (C) 1983, 1985, 1986, 1987 by
* Digital Equipment Corporation, Maynard, Mass.
*
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by Digital Equipment
* Corporation.
*
* Digital assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by Digital.
*

IDENTIFICATION DIVISION.
PROGRAM-ID. COBREC.

*****************************************************************
*                                                               *
*    This is the receive program of the DECnet intertask        *
*    communication example programs for COBOL.                  *
*                                                               *
*    To task build, use the following command string:           *
*                                                               *
*   COBREC,COBREC=COBREC,[1,1]NETFOR/LB,C81LIB/LB,RMSLIB/LB     *
*   /                                                           *
*   UNITS=10                                                    *
*   ACTFIL=4                                                    *
*   EXTTSK=1000                 (if RMS is included)            *
*   //                                                          *
*                                                               *
*   This program illustrates the use of a long connect block.   *
*                                                               *
*****************************************************************

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT DUMMY-FILE ASSIGN TO "COBREC.DUM".

DATA DIVISION.
FILE SECTION.
FD      DUMMY-FILE
        LABEL RECORD STANDARD.
01      DUMMY-FILE-REC.
```

(continued on next page)

```
   02   FILLER          PIC X(132).

WORKING-STORAGE SECTION.
01  ARRAYS.
    03  IOST.
        05  IOSTAT OCCURS 2 TIMES PIC S9999 USAGE COMP.
    03  MSTAT.
        05  MSTATS OCCURS 3 TIMES PIC S9999 USAGE COMP.
    03  IOST1.
        05  IOSTAT1 OCCURS 2 TIMES PIC S9999 USAGE COMP.
    03  IOST2.
        05  IOSTAT2 OCCURS 2 TIMES PIC S9999 USAGE COMP.
01  STORE-STUFF.
    03  OPNLUN          PIC 99       COMP VALUE 2.
    03  MBXFLG          PIC 99       COMP VALUE 1.
    03  MLTYP           PIC 9        USAGE COMP.
    03  MLSIZ           PIC 99       COMP VALUE 98.
    03  MLBOX           PIC X(98).
    03  INDX            PIC 99       USAGE COMP.
    03  ACCLUN          PIC 99       COMP VALUE 3.
    03  RECSIZ          PIC 99       COMP VALUE 50.
    03  RECBUF          PIC X(50).
    03  NUMMES          PIC 99       COMP VALUE 0.
    03  INTSIZ          PIC 9        COMP VALUE 6.
    03  INTMES          PIC X(6).
    03  SNDSIZ          PIC 99       COMP VALUE 10.
    03  SNDDAT          PIC X(10).
PROCEDURE DIVISION.

*****************************************************************
*                                                               *
*       Access the network.  If the call completes              *
*       unsuccessfully, exit.                                   *
*                                                               *
*****************************************************************

A100-START.
        CALL "OPNNTW" USING
                OPNLUN
                IOST
                MSTAT
                0
                0
                MBXFLG.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                GO G100-END.
        IF MSTATS (1) = 0 GO C100-CLOSNET.

*****************************************************************
*                                                               *
*       Check to see if there is anything on the task's         *
*       data queue.                                             *
*                                                               *
*****************************************************************
```

```
B100-NETDAT.
        CALL "GNDNT" USING
                IOST1
                MLTYP
                MLSIZ
                MLBOX.

*****************************************************************
*                                                               *
*       Wait for completion of a GNDNT or RECNT call.  If a     *
*       RECNT call completes (INDEX = 2), process a receive.    *
*       If a GNDNT call completes unsuccessfully, close the     *
*       network and exit.  If the type of data message in       *
*       the mailbox is not a connect request or an interrupt    *
*       message, close the network and exit.  If an interrupt   *
*       message is in the mailbox (MLTYP = 2), simply issue     *
*       a new GNDNT.                                            *
*                                                               *
*****************************************************************

B110-WAIT.
        CALL "WAITNT" USING
                INDX
                IOST1
                IOST2.
        IF INDX = 2 GO D100-INQREC.
        IF IOSTAT1 (1) NOT = 1 GO C100-CLOSNET.
        IF MLTYP NOT < 3 GO C100-CLOSNET.
        IF MLTYP = 2 GO B100-NETDAT.

*****************************************************************
*                                                               *
*       A connect request is in the mailbox.  Accept the        *
*       request to establish a logical link.  If the call       *
*       completes unsuccessfully, issue a new GNDNT.            *
*                                                               *
*****************************************************************

        CALL "ACCNTW" USING
                ACCLUN
                IOST
                MLBOX.
        IF IOSTAT (1) NOT = 1 GO B100-NETDAT.

*****************************************************************
*                                                               *
*       Pick up the data from the transmitting task.  Issue     *
*       a new GNDNT and wait for completion.                    *
*                                                               *
*****************************************************************

        CALL "RECNT" USING
                ACCLUN
```

(continued on next page)

```
                IOST2
                RECSIZ
                RECBUF.
        GO B100-NETDAT.

*****************************************************************
*                                                               *
*       A disconnect or abort was received.  Deaccess the       *
*       network and exit.                                       *
*                                                               *
*****************************************************************

C100-CLOSNET.
        CALL "CLSNTW".
        GO G100-END.

*****************************************************************
*                                                               *
*       An inquiry was received.  Increment the message         *
*       count.  If the call completed unsuccessfully, send      *
*       an interrupt message containing the message number      *
*       in which the error occurred.                            *
*                                                               *
*****************************************************************

D100-INQREC.
        ADD 1 TO NUMMES.
        IF IOSTAT2 (1) = 1 GO E100-SEND.
        MOVE NUMMES TO INTMES.
        CALL "XMINT" USING
                ACCLUN
                IOST
                INTSIZ
                INTMES.
        GO F100-REC.

*****************************************************************
*                                                               *
*       Send data to the task.                                  *
*                                                               *
*****************************************************************

E100-SEND.
        CALL "SNDNTW" USING
                ACCLUN
                IOST
                SNDSIZ
                SNDDAT.

*****************************************************************
*                                                               *
*       Issue a new RECNT and wait for completion.              *
*                                                               *
*****************************************************************
```

```
F100-REC.
        CALL "RECNT" USING
                ACCLUN
                IOST2
                RECSIZ
                RECBUF.
        GO B110-WAIT.
G100-END.
        DISPLAY "COBREC -- END OF EXECUTION".
        STOP RUN.
```

3.8.21 BASIC–PLUS–2 Intertask Communication Programming Examples

The following two programs are examples of BASIC–PLUS–2 intertask communication. They are cooperating tasks. BASTRN is a transmit task; BASREC is a receiver task.

These examples are included in your tape or disk kit.

3.8.21.1 Transmit Example

The following program, BASTRN, accesses the network, connects to BASREC, transmits inquiries to BASREC, and processes responses from BASREC. When the program completes sending inquiries, it disconnects the link, stops accessing the network, and exits.

```
!
! Copyright (C) 1983, 1985, 1986, 1987 by
! Digital Equipment Corporation, Maynard, Mass.
!
!
! This software is furnished under a license and may be used and  copied
! only  in  accordance  with  the  terms  of  such  license and with the
! inclusion of the above copyright notice.  This software or  any  other
! copies  thereof may not be provided or otherwise made available to any
! other person.  No title to and ownership of  the  software  is  hereby
! transferred.
!
! The information in this software is subject to change  without  notice
! and  should  not  be  construed  as  a commitment by Digital Equipment
! Corporation.
!
! Digital assumes no responsibility for the use or  reliability  of  its
! software on equipment which is not supplied by Digital.
!


10      !!!                                                         !!! &
        !!!     BASTRN.B2S - Transmit inquiries to BASREC and       !!! &
        !!!                  process responses                      !!! &
        !!!                                                         !!! &
        !!!    To task build, edit the task build command file      !!! &
        !!!    and the ODL file created by the build.               !!! &
        !!!                                                         !!! &
        !!!     >Add the line                                       !!! &
        !!!             ACTFIL=4                                    !!! &
        !!!     to the task build command file.                     !!! &
        !!!                                                         !!! &
        !!!     >Append                                             !!! &
        !!!             -NETLIB                                     !!! &
        !!!     to the USER: line of the ODL file.                  !!! &
        !!!                                                         !!! &
        !!!     >Add the line                                       !!! &
        !!!             NETLIB: .FCTR LB:[1,1]NETFOR/LB             !!! &
        !!!     to the ODL file.                                    !!! &

        !!! Define array constants !!!                                 &
        DIM IOST%(1%),MSTAT%(2%)        !Define array elements        &
 \      ERRMES$=STRING$(2%,0%)          !Define max string length     &
 \      CONBLK$=STRING$(72%,0%)         !STRING$(LENGTH,ASCII VALUE)  &
 \      RECBUF$=STRING$(10%,0%)         !                             &
 \      SNDBUF$=STRING$(50%,0%)         !

20      INPUT "Node-name <MASTER>";NDNAM$ \ IF NDNAM$="" THEN         &
                NDNAM$="MASTER" ELSE IF LEN(NDNAM$)>6% THEN           &
                PRINT "Node-name too long, please re-enter"           &
 \              PRINT \ GOTO 20
```

(continued on next page)

```
30      INPUT "Receive task-name <RECVER>";TSKNAM$ \ IF TSKNAM$=""      &
                THEN TSKNAM$="RECVER" ELSE IF LEN(TSKNAM$)>6% THEN      &
                PRINT "Task-name too long, please re-enter"             &
 \              PRINT \ GOTO 30


40      !!! Define constants !!!                                        &
        IMMED%=-1%                      !Set IMMED% to true for GNDNTW  &
 \      OPNLUN%=1%                      !Network OPNNT LUN              &
 \      CONLUN%=2%                      !CONNT LUN for the logical link &
 \      XMITS%=20%                      !The number of inquiries        &
                                        ! to send to the remote node    &
 \      SNDSIZ%=50%                     !The size of the messages to    &
                                        ! send to the remote node       &
 \      RECSIZ%=10%                     !The size of the messages to    &
                                        ! receive                       &
 \      NDNAM.LEN%=LEN(NDNAM$)          !Length of the node-name        &
 \      TSKNAM.LEN%=LEN(TSKNAM$)        !Length of the task-name

50      !!! Access the network !!!                                      &
        CALL     OPNNTW BY REF(OPNLUN%,IOST%(),MSTAT%())                &
 \      IF      IOST%(0%)=1% THEN 60    !If successful, build the       &
                                        !  connect block                &
        ELSE    PRINT "Network OPEN failed, IOST=";IOST%(0%);IOST%(1%)  &
 \      GOTO    160                     !Open failed. Print the status  &
                                        !  block and exit

60      !!! Build a Format 1 connect block !!!                          &
        CALL    BFMT1 BY REF(STAT%,CONBLK$,NDNAM.LEN%,NDNAM$            &
                ,DUMMY%,TSKNAM.LEN%,TSKNAM$)                            &
 \      IF      STAT% THEN 70 ELSE      !If success go on               &
        PRINT "Error building connect block"                            &
                                        !Else type out an error message &
 \      GOTO    150                     !  and exit

70      !!! Connect to the task on the remote node !!!                  &
        CALL    CONNTW BY REF(CONLUN%,IOST%(),CONBLK$)                  &
 \      IF      IOST%(0%)=1% THEN 80    !If success, tell it            &
        ELSE    PRINT "Connect Fail: IOST=";IOST%(0%);",";IOST%(1%)     &
                                        !Else print status block        &
 \      GOTO    150                     !Else print status block and exit

80      PRINT   "Link enabled"          !Print connect confirmation     &
                                        !  to network

90      !!! Send and receive messages to and from the remote node !!!   &
        FOR     MESNUM%=1% TO XMITS%

100             !!! First get any error messages sent from the other !!!&
                !!! side via interrupt messages !!!                     &
                IF MSTAT%(0%)=0% THEN 110 !If MSTAT%(0%)=0% no messages &
                                          !  are there                  &
                ELSE CALL GNDNTW BY REF(IOST%(),MLTYP%,2%,ERRMES$       &
                        ,DUMMY%,IMMED%,2%) !Get the message             &
 \              IF IOST%(0%)<>1% THEN 110 !If we couldn't get message   &
```

```
                                    !  just ignore it                &
                ELSE PRINT "Error on inquiry #";ASCII(LEFT(ERRMES$,1%))
                                    !Print out the message

110             !!! Send the inquiry !!!                              &
                CALL SNDNTW BY REF(CONLUN%,IOST%(),SNDSIZ%,SNDBUF$)   &
 \              IF IOST%(0%)=1% THEN 120!If success continue          &
                ELSE PRINT "Error on inquiry during send: ";MESNUM%   &
                                    !Otherwise type out an error      &
 \              GOTO 130             !message and start a new message

120             !!! Receive the response from the remote node !!!     &
                CALL RECNTW BY REF(CONLUN%,IOST%(),RECSIZ%,RECBUF$)   &
 \              IF IOST%(0%)=1% THEN 130!If success continue          &
                ELSE PRINT "Error on inquiry during receive: ";MESNUM% &
                                    !Otherwise type out an            &
                                    !  error message

130     NEXT    MESNUM%             !End of loop

140     !!! Disconnect the link !!!                                   &
        PRINT   "Link disabled"     !Print out disconnect message     &
 \      CALL    DSCNTW BY REF(CONLUN%,IOST%())

150     !!! Come here to deaccess the network and exit !!!            &
        CALL    CLSNTW

160     PRINT   "End of execution"                                    &
 \      END
```

3.8.21.2 Receive Example

BASREC receives inquiries from BASTRN. It returns any errors to BASTRN as interrupt messages, which BASTRN displays on the terminal.

```
!
! Copyright (C) 1983, 1985, 1986, 1987 by
! Digital Equipment Corporation, Maynard, Mass.
!
!
! This software is furnished under a license and may be used and copied
! only in accordance with the terms of such license and with the
! inclusion of the above copyright notice. This software or any other
! copies thereof may not be provided or otherwise made available to any
! other person. No title to and ownership of the software is hereby
! transferred.
!
! The information in this software is subject to change without notice
! and should not be construed as a commitment by Digital Equipment
! Corporation.
!
! Digital assumes no responsibility for the use or reliability of its
! software on equipment which is not supplied by Digital.
!

10      !!!                                                         !!! &
        !!!     BASREC.B2S - Receive inquiries from BASTRN and      !!! &
        !!!                  send back responses                    !!! &
        !!!                                                         !!! &
        !!!    To task build, edit the task build command file      !!! &
        !!!    and the ODL file created by the build.               !!! &
        !!!                                                         !!! &
        !!!     >Add the line                                       !!! &
        !!!             ACTFIL=4                                    !!! &
        !!!     to the task build command file.                     !!! &
        !!!                                                         !!! &
        !!!     >Append                                             !!! &
        !!!             -NETLIB                                     !!! &
        !!!     to the USER: line of the ODL file.                  !!! &
        !!!                                                         !!! &
        !!!     >Add the line                                       !!! &
        !!!             NETLIB: .FCTR LB:[1,1]NETFOR/LB             !!! &
        !!!     to the ODL file.                                    !!! &

        !!! Initialize constants !!!                                   &
        DIM MSTAT%(2%),IOST%(1%),IOST1%(1%),IOST2%(1%)                &
 \      INTMES$=STRING$(2%,0%)          !Define max length of strings &
 \      MLBX$=STRING$(98%,0%)           !STRING$(LENGTH,ASCII VALUE)  &
 \      RECBUF$=STRING$(50%,0%)         !                             &
 \      SNDDAT$=STRING$(10%,0%)         !

20      !!! More constants !!!                                        &
        OPNLUN%=1%                      !Network OPNNT LUN            &
 \      ACCLUN%=2%                      !Accnt LUN for logical link   &
 \      RECSIZ%=50%                     !Size of data buffer to be    &
                                        !  received                   &
 \      INTSIZ%=2%                      !Size of interrupt data buffer &
                                        !  to send                    &
```

```
\       NUMMES%=0%                          !Number of messages received    &
\       INDEX=0%                            !Receive completion flag        &
\       SNDSIZ%=10%                         !Number of bytes to send back

30      !!! Access network !!!                                              &
        CALL    OPNNTW BY REF(OPNLUN%,IOST%(),MSTAT%())                     &
\       IF      IOST%(0%)<>1% THEN 140  !If failure just exit               &
        ELSE    IF MSTAT%(0%)=0% THEN 90!If nothing on mailbox              &
                                        !  just close and exit

40      CALL    GNDNT BY REF(IOST1%(),MLTYP%,98%,MLBX$)                     &
                                        !Issue Get Network Data

50      CALL    WAITNT BY REF(INDEX%,IOST1%(),IOST2%())                     &
                                        !Wait for a completion              &
\       IF      INDEX%=2% THEN 100      !If INDEX%=2% then a receive        &
                                        !  has been completed

60      !!! Network data has been received !!!                              &
        IF      IOST1%(0%)<>1% THEN 90  !If GNDNT failed, just              &
                                        !  close and exit                   &
        ELSE    IF MLTYP%>=3% THEN 90   !If MLTYP%>=3% then link has        &
                                        !  been broken                      &
        ELSE    IF MLTYP%=2% THEN 40    !If MLTYP%=2% we've received        &
                                        !  an interrupt message. Just       &
                                        !  issue a GNDNT

70      !!! We've received a connect request - issue an accept !!!          &
        CALL    ACCNTW BY REF(ACCLUN%,IOST%(),MLBX$)                        &
\       IF      IOST%(0%)<>1% THEN 40   !If failure issue a new GNDNT

80      !!! Issue a receive to pick up data !!!                             &
        CALL    RECNT BY REF(ACCLUN%,IOST2%(),RECSIZ%,RECBUF$)              &
\       GOTO    40                      !Issue a new GNDNT and              &
                                        !  wait for the completion

90      !!! We come here upon receiving a disconnect or abort !!!           &
        CALL    CLSNTW                  !Deaccess the network               &
\       GOTO    140                     !  and exit

100     !!! We come here if we receive an inquiry !!!                       &
        NUMMES%=NUMMES%+1%              !Increment the message count        &
\       IF      IOST2%(0%)=1% THEN 120  !If IOST2%(0%)=1 all's okay

110     !!! If there was an error, send back an interrupt message !!!       &
        !!!   with message number                                  !!!     &
        INTMES$=CHR$(NUMMES%)+CHR$(0%)  !Send the message number            &
\       CALL    XMINT BY REF(ACCLUN%,IOST%(),INTSIZ%,INTMES$)               &
\       GOTO    130                     !Go issue a new receive             &

120     !!! Here the user can look at the data received in RECBUF$ !!!      &
        !!! and respond by replacing the requested information     !!!      &
        !!! into SNDDAT$                                          !!!      &
        !!! Send back the data and issue a RECNT                  !!!      &
        CALL    SNDNTW BY REF(ACCLUN%,IOST%(),SNDSIZ%,SNDDAT$)
```

(continued on next page)

```
130     CALL    RECNT BY REF(ACCLUN%,IOST2%(),RECSIZ%,RECBUF$)          &
 \      GOTO    50                      !Wait for a completion

140     !!! Exit program !!!                                            &
        PRINT   "End of program execution"                              &
 \      END
```

3.9 Remote File Access

This section contains descriptions and usage guidelines specific to the remote file access calls listed alphabetically in Table 3–3.

Table 3–3: Remote File Access Calls

Call	Function
ACONFW	Set record and file access options
ATTNFW	Set extended attributes
CLSNFW	Close a file
DELNFW	Delete a file
EXENFW	Execute a file
GETNFW	Read a single record
OPANFW	Open and append a sequential file
OPMNFW	Open and modify a sequential file
OPRNFW	Open and read a sequential file
OPUNFW	Open and update a sequential file
OPWNFW	Create, open, and write a sequential file
PRGNFW	Discard an open file
PUTNFW	Write a record to a file
RENNFW	Rename a file
SPLNFW	Open, write, and print a file
SUBNFW	Open, write, and execute a file

These calls are implemented by subroutines. The network open call, OPNNT[W], and the network close call, CLSNT[W], are also used in remote file access operations. You must always issue OPNNT[W] first because OPNNT[W] lets your task access the network. Issue CLSNT[W] last to close your task's access to the network.

3.9.1 Opening Files

The following nine subroutines open files:

ACONFW	Specifies record and file access options before performing a specific file operation.
ATTNFW	Specifies extended attributes before performing OPWNFW, SPLNFW, SUBNFW, OPRNFW, OPANFW, and RENNFW calls.
OPRNFW	Opens an existing file for reading, beginning with the first record.
OPANFW	Opens an existing file and appends records to the end of the file.
OPMNFW	Opens an existing file for record modification.
OPUNFW	Opens an existing file for record update.
OPWNFW	Creates and opens a file, then writes records to it, beginning with the first record position.
SPLNFW	Performs the same function as OPWNFW and then prints the file.
SUBNFW	Performs the same function as OPWNFW and then executes the file.

Each open subroutine creates a DECnet logical link to the node where the file resides and then creates and opens the file. You must use the same LUN to open, write, and close the file. This LUN must not be in use.

Issue an ATTNFW or ACONFW call immediately before OPRNFW, OPANFW, and RENNFW calls to specify additional attributes to be returned after the open operation completes.

3.9.2 Performing File Operations

The following subroutines perform file operations:

EXENFW	Executes a remote file.
DELNFW	Deletes a remote file.
RENNFW	Renames a remote file.

3.9.3 Performing Record Operations

The following subroutines perform record operations:

GETNFW Reads a record from a remote file.

PUTNFW Writes a record to a remote file.

3.9.4 Closing Files and Completing Calls

When you complete a file access operation, use CLSNFW to close the file. To clean up errors before closing the file, use PRGNFW for your close operation. Both CLSNFW and PRGNFW disconnect the logical link and free the logical unit number for use. If you do not perform a close operation before attempting a CLSNT[W] to stop accessing the network, or if a network abort occurs while the file is open, the network closes the file. However, all data may not have been transferred successfully.

Remote file access calls are synchronous and do not return to the user until an operation completes.

3.9.5 Setting Task Build Parameters

DECnet–RSX uses network file access routines (NFARs) as the local node interface to access remote files for user applications. At task build time you can override defaults to tailor these NFARs for a particular application. You can set the following task build parameters:

- Event flags .TREF and .RCEF. (The defaults are 17 and 18, respectively.)
- Buffering level. (The default is 2.)
- Maximum record size. (The default is 256. bytes.)
- Buffer space allocation. (There is no default.)

3.9.5.1 Setting Event Flags

The network file access routines (NFARs) require the exclusive use of two event flags. The default event flags are 17(.TREF) and 18(.RCEF). To override these defaults, issue the following commands in the task builder command file:

```
GBLDEF=.TREF:value
```

```
GBLDEF=.RCEF:value
```

The *value* variable specifies an event flag and must be in the form of an octal integer from 1 to 200 (octal). Event flags 33. through 64. are global flags.

3.9.5.2 Setting Buffering Level

The NFARs can be configured for multibuffering to improve throughput. This requires more internal buffering space than the default buffering level of 2. To override this default, issue the following command in the task builder command file:

```
GBLDEF=$NFNMB:buffering-level
```

The *buffering-level* variable specifies an integer from 1 to 4. For an RSX or IAS remote system, use the buffering level that the remote system uses. Ask the system manager for the information.

3.9.5.3 Setting Maximum Record Size

The internal buffers used by the NFARs must be large enough to hold the largest data record in the remote file. The default maximum record size is 256 bytes. To override this default, include the following command in the task builder command file:

```
GBLDEF=$NFRSZ:record-size
```

The *record-size* variable specifies an octal value. For an RSX or IAS remote system, use the record size that the remote system uses.

In calculating the maximum record size, note that certain file types require extra bytes, as follows:

- ASCII files require 2 extra bytes for carriage return and line feed characters that are appended to each ASCII record.
- Sequenced variable length records require 2 extra bytes for the sequence number included with each record.
- ASCII files with sequenced variable length records require 4 extra bytes.

3.9.5.4 Setting Buffer Space Allocation

The NFARs allocate buffer space from the file storage region used by the File Control System (FCS–11). This space is allocated in the P-section $$FSR1. Your task must include the module NFAFSR from the NETFOR object library. Be sure to enter the following line in the task build command as an input file:

```
[1,1]NETFOR/LB:NFAFSR
```

Use the following formula to calculate the required buffer space for performing remote file access:

$$(((\$NFRSZ + 14.)*(\$NFNMB + 1)) + 64.)*(max\text{-}rem\text{-}files) + (space.* < max\text{-}loc\text{-}files >)$$

where

space is the space overhead per local file.

max-rem-files is the maximum number of remote files that can be open concurrently.

max-loc-files is the maximum number of local files that can be open concurrently.

The *space* variable for local file overhead value is 512. if your language uses FCS–11. If your language uses a different file system, refer to its documentation for the exact value.

At task build time, use the value that you calculated with the formula to extend the file storage region. Include the following command in the task builder command file, entering the value in octal bytes:

```
EXTSCT=$$FSR1:value
```

3.9.5.5 Using the Task Build Procedure

A task must link to NETFOR.OLB to use the DECnet–RSX remote file access capabilities. Edit the ODL file that the compiler created. The following example shows a task using the CMD and ODL files. This task uses the defaults for the buffer size ($NFRSZ), the number of buffers ($NFNMB), and only one link. The underlined items indicate the required edits for the task builder to include remote file access capabilities in the task. Boldface items are required for network access.

FORTRAN Example:

```
FILES.CMD {.B SY:FILES,SY:FILES/-SP=SY:FILES.LB:[1,1]F4POTS
                    LB:[1,1]NETFOR/LB,NETFOR/LB:NFAFSR
/
EXTSCT=$$FSR1:2700
//
```

You enter:

```
MCR>TKB @FILES (RET)
```

COBOL Example:

```
FILES.CMD
     SY:FILES,SY:FILES/-SP=SY:FILES/MP
     EXTSCT=$$FSR1:2700
     //
FILES.ODL

;MERGED ODL FILE CREATED ON 26-FEB-82 AT 16:54:32
;COBOL STANDARD ODL FILE GENERATED ON 26-FEB-82 16:46:29
;COBOBJ=FILES.OBJ
;COBMAIN
LIBR1:      .FCTR    LB:[1,1]NETFOR/LB-NETFOR/LB:NFAFSR
CBOBJ$:     .FCTR    SY:[200,200]FILES
CBOTS$:     .FCTR    LB:[1,1]COBLIB/LB
OBJRT$:     .FCTR    CBOBJ$-CBOTS$-LIBR1
            .ROOT    OBJRT$
            .END
```

You enter:

```
MCR>TKB @FILES (RET)
```

BASIC–PLUS–2 Example:

```
FILES.CMD
     SY:FILES/CP/FP,FILES/-SP=SY:FILES/MP
     UNITS = 14
     ASG = TI:13
     ASG = SY:5:6:7:8:9:10:11:12
     EXTSCT = $$FSR1:2700
     //

FILES.ODL
              .ROOT   BASIC2-RMSROT-USER,RMSALL
USER:         .FCTR   SY:FILES-NETLIB
LIBR:         .FCTR   LB:[1,1]BP2OTS/LB
NETLIB:       .FCTR   LB:[1,1]NETFOR/LB-NETFOR/LB:NFAFSR
@LB:[1,1]BP2IC1
@LB:[1,1]RMS11X
              .END
```

You enter:

```
MCR>TKB @FILES (RET)
```

3.9.6 Using ASCII Zero (ASCIZ) Strings

Some of the network file access subroutines require that you provide one or more arguments in the CALL statement as ASCIZ strings. An ASCIZ string is a string of ASCII characters terminated by a binary (0).

You can create an array/numeric data item, store the string in the array/numeric data item, and then set the last element to zero (0).

FORTRAN Example:

```
   DIMENSION IFILE (12)
   DATA IFILE/'DK','0:','[1',',4',']C',
   'N','TR','OL','.A','LG',';2'/
   IFILE(12)=0
```

You then specify the array name in the CALL statement:

```
CALL OPRNFW (lun,status,node,,IFILE)
```

COBOL Example:

```
   01  NULL1 PIC 9 COMP VALUE 0.

   01  NULLS REDEFINES NULL1.

   03  NULL OCCURS 2 TIMES PIC X(1).

01  IFILE PIC X(23) VALUE "DK0:[200,200,]NAME.CBL;1"
    .
    .
    .
STRING IFILE
    NULL (1)

    INTO IDENT.
```

You then specify the string in the CALL statement:

```
CALL "OPRNFW" USING lun,status,node,ident.
```

BASIC-PLUS-2 Example:

```
IFILE$="DK0:[200,200]NAME.B2S;1"+CHAR$(0%)CHAR$(0%)
```

You then specify the array name in the CALL statement:

```
CALL OPRNFW BY REF (lun%,status%(),node$,ident$,ifile$)
```

3.9.7 Common Argument Definitions for Remote File Access Calls

This section defines the common arguments for remote file access calls. The general group defines arguments common to all languages, and three individual groups define arguments specific to FORTRAN, COBOL, and BASIC–PLUS–2.

GENERAL

- *lun*

  is an integer variable or constant that specifies the logical unit number of the logical link created for a specific file access operation.

- *node*

  specifies the name of the target node. It is a 1- to 7-element array/string that ends with a binary 0 and contains a 1- to 6-character ASCIZ string.

- *ident*

  contains three successive ASCIZ strings: the user ID, password, and account number necessary to access remote node files. It is a 1- to 72.-element ASCIZ array.

  Enter a null value (0) for each item not required by the remote node or each item previously entered. For example, you may have already entered the required information in an alias node name block.

- *ifile*

  is a byte array/string containing a variable length ASCIZ string that contains the file specification for a file access operation. Be sure to use the remote node's file specification syntax.

FORTRAN

- References to integers imply single-precision integer values.

- *status*

  specifies an array that contains completion status information on return from the call. This 2-element single-precision integer array contains the following values when the call completes:

status(1)	returns a completion code. A positive 1 indicates success; a negative integer indicates an error. Appendix C lists and describes the error codes.

status(2) depends on the contents of the first status word. Refer to Appendix C.

COBOL

- For DECnet COBOL, the logical unit number 1 is a reserved number and should never be assigned for the *lun* argument.

- *status*

  specifies an elementary numeric data item that contains completion status information on return from the call. This elementary numeric data item contains the following values when the call completes:

  status(1) returns a completion code. A positive 1 indicates success; a negative integer indicates an error. Appendix C lists and describes the error codes.

  status(2) depends on the content of the first status word. Refer to Appendix C.

BASIC–PLUS–2

- *status*%( )

  specifies an array that contains completion status information on return from the call. This 2-element integer array contains the following values when the call completes:

  status%(0) returns a completion code. A positive 1 indicates success; a negative integer indicates an error. Appendix C lists and describes the error codes.

  status%(1) depends on the contents of the first status word. Refer to Appendix C.

ACONFW
(Set Access Options)

3.9.8 ACONFW — Set Access Options

Use:

Call ACONFW before a specific file operation to specify record and file access options to apply to that file operation. These options remain in effect until the file is closed.

Formats:

FORTRAN: CALL ACONFW (*lun*,*status*,[*fac*],[*shr*],[*fop*])

COBOL: CALL "ACONFW" USING *lun*,*status*,[*fac*],[*shr*],[*fop*].

BASIC: CALL ACONFW BY REF (*lun*%,*status*%( ), [*fac*%],[*shr*%],[*fop*%( )])

Arguments:

lun

specifies the logical unit number of the logical link assigned to the file operation for which to set options.

* *status*

specifies completion status on return from ACONFW. See the definitions for your language in Section 3.9.7. Refer to Appendix C for error code descriptions.

fac

specifies the operations to allow during file access. Use this argument only for open and create operations. The *fac* value overrides the type of access that the OP*x*NFW call specifies. Valid values are:

1	Put access
2	Get access (default)
4	Delete record access
10	Update record access
20	Truncate file access

40	Block I/O
41	Block I/O write
42	Block I/O read

shr

specifies the file sharing to be allowed by the remote system. Use this argument only for open and create operations. The actual functioning depends on the remote system's capabilities. Valid values are:

1	Put access
2	Get access (default)
4	Delete record access
10	Update record access
100	No access to others

* *fop*

specifies a 3-word array for file-processing options that open, create, and close operations will use. To specify a *fop* value for close operations, first open the file, and then make the ACONFW call, because the open call overwrites the current value. Valid values are:

First word:

400	Create contiguous file
1000	Supersede existing file
4000	Create temporary file
10000	Create temporary file and mark for delete

Second word:

40	Maximize version number on create
100	Spool on close
400	Execute on close
1000	Delete on close
20000	Truncate on close

Completion of open/create operations returns:

First word:

40	File is a directory (system dependent)
100	File is locked
400	File is contiguous

ATTNFW
(Set Extended Attributes)

3.9.9 ATTNFW — Set Extended Attributes

Use:

Call ATTNFW to specify extended attributes to use with a create, open, or close file operation. You call ATTNFW immediately before the operation; the attributes are returned on completion of the operation.

Call ATTNFW with the following operations:

Operation	Calls	What ATTNFW Does
Create	OPWNFW SPLNFW SUBNFW	Specifies additional attributes in creating the file.
Open	OPRNFW OPANFW RENNFW	Specifies additional attributes to be returned when opening the file.
Close	CLSNFW	Specifies a change-attributes-on-close sequence.

Formats:

FORTRAN: CALL ATTNFW (*lun,status,*[*namesize*],[*name*],[*atb*],
[*protblk*],[*owner*],[*dateblk*])

COBOL: CALL "ATTNFW" USING *lun,status,*[*namesize*],[*name*],[*atb*],
[*protblk*],[*owner*],[*dateblk*].

BASIC: CALL ATTNFW BY REF (*lun*%,*status*%( ),[*namesize*%],[*name*$],
[*atb*%( )],[*protblk*%( )],
[*owner*$],[*dateblk*%( )])

Arguments:

lun

specifies the logical unit number of the logical link for the file operation.

* *status*

specifies completion information on return from ATTNFW. See the definition for your language in Section 3.9.7. Refer to Appendix C for error code descriptions.

namesize

specifies the maximum length of the array/string that can be returned to the resultant file specification. Use this single-word argument for open and create operations.

* *name*

specifies the array/string containing the resultant file name. This argument can be used for all operations. The returned file name is an ASCIZ string.

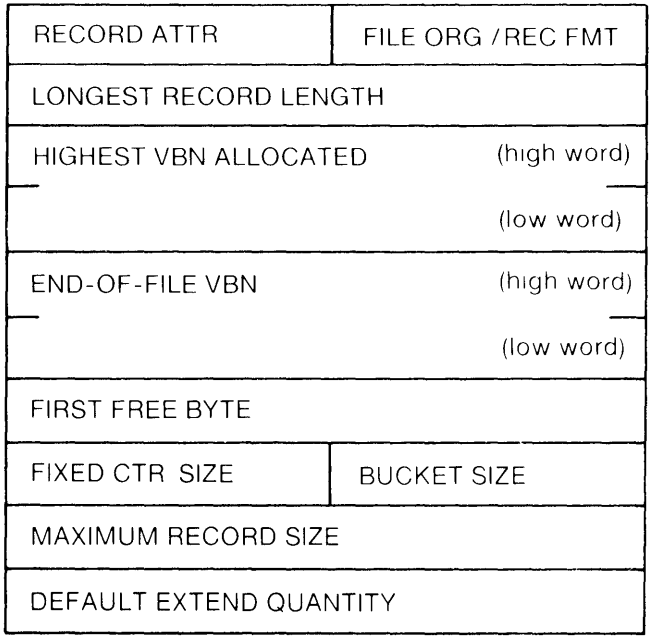
* *atb*

specifies a Files-11 user file attributes block. When specifying create operations, the user program is responsible for setting valid values because the NFARs do not check these values.

NOTE

If *atb* is specified, the *ichar*(2), *ichar*(3), and *len* arguments are ignored when used with open or create calls. Use the fields NF$ORG, NF$RAT, and NF$MRS, instead.

When specifying create and open operations, *atb* returns a 10.-word block in the following format.

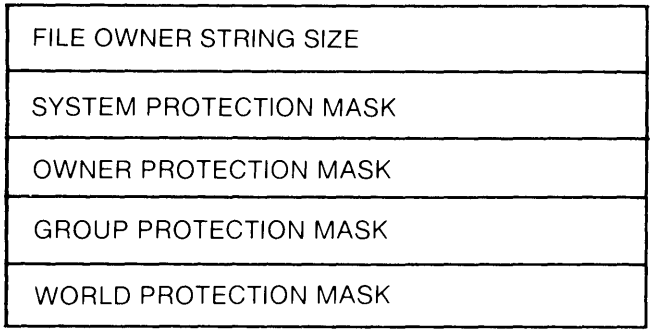


LKG-1036-87

For further information on the values of these fields, see the *RSX-11M/M-PLUS* or *Micro/RSX I/O Operations Reference Manual*.

* *protblk*

specifies an array containing file protection information to be used as input for create and close operations, and returns information from create and open operations. The following format describes a 5-word array.



LKG-1037-87

If the file owner size is 0, the owner string is not returned.

The format of the protection masks is as follows:

Bit 0 = Deny read access
Bit 1 = Deny write access
Bit 2 = Deny execute access
Bit 3 = Deny delete access
Bit 4 = Deny append access
Bit 5 = Deny directory list access
Bit 6 = Deny update access
Bit 7 = Deny change protection access
Bit 8 = Deny extend access

If a word is set to −1, that protection is not sent.

* *owner*

  specifies an ASCIZ string/array identifying the file owner to use as input for create operations and for output from create and open operations. The first word of the *protblk* array must specify the maximum size of this string/array.

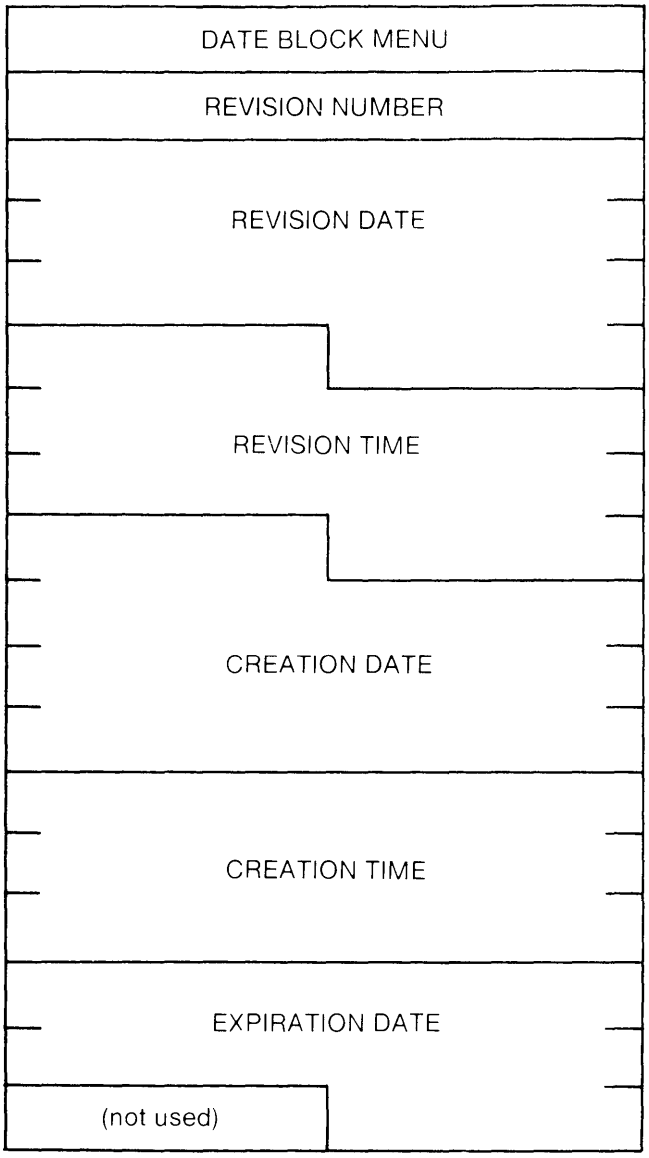
* *dateblk*

  specifies a 19-word array whose fields contain the file's revision number, creation and revision date and time, and/or expiration date. The date block menu specifies the fields in the block that create and close operations can get

values from, and create and open operations can return values to. Date block menu values are as follows:

Bit 0 = Revision number
Bit 1 = Revision date and time
Bit 2 = Creation date and time
Bit 3 = Expiration date

The date block fields contain Files-11 time stamps. The time stamps are in ASCII and have the format *ddmmmyyhhmmss*, where *mmm*, the month abbreviation, is in uppercase letters. Leading zeros are included. This is the format of the date block.



LKG-1038-87

For further information on the format of dates in a block, see the *RSX-11M/M-PLUS* or *Micro/RSX I/O Operations Reference Manual*.

CLSNFW
(Close a File)

3.9.10 CLSNFW — Close a File

Use:

Call CLSNFW to close a remote file. CLSNFW forces completion of all pending file operations, ensures that the file directory information is valid, and optionally modifies the attributes that the *changeattr* argument specifies. The logical unit number is freed when the CLSNFW call completes.

Note that some systems do not let you change attributes on close.

Formats:

FORTRAN: CALL CLSNFW (*lun*,*status*,[*changeattr*])

COBOL: CALL "CLSNFW" USING *lun*,*status*,[*changeattr*].

BASIC: CALL CLSNFW BY REF (*lun*%,*status*%( ),[*changeattr*%])

Arguments:

lun

specifies the logical unit number of the logical link to close. See the definition in Section 3.9.7. Use the same LUN that the previous open call specified.

* *status*

specifies completion status information on return from CLSNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

changeattr

specifies attributes to change when this file closes. A previous ATTNFW call must specify these attributes either at open time or just before this call. Valid values are:

2 Change protection
4 Change dates and times

DELNFW
(Delete a File)

3.9.11 DELNFW — Delete a File

Use:

Call DELNFW to delete a remote file.

Formats:

FORTRAN: CALL DELNFW (*lun,status,node,ident,ifile*)

COBOL: CALL "DELNFW" USING *lun,status,node,ident,ifile.*

BASIC: CALL DELNFW BY REF(*lun*%,*status*%( ),*node*$,*ident*$,*ifile*$)

Arguments:

lun

specifies the logical unit number of the logical link to delete. See the definition in Section 3.9.7.

* *status*

specifies completion status information on return from DELNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file to delete. See the definition in Section 3.9.7.

ident

is an array/string containing explicit access control information. See the definition in Section 3.9.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be deleted. See the definition in Section 3.9.7.

EXENFW
(Execute a File)

3.9.12 EXENFW — Execute a File

Use:

Call EXENFW to submit an existing remote file to the batch or command file processor. The remote file is not deleted after this call completes.

Formats:

FORTRAN: CALL EXENFW (*lun,status,node,ident,ifile*)

COBOL: CALL "EXENFW" USING *lun,status,node,ident,ifile.*

BASIC: CALL EXENFW BY REF (*lun*%,*status*%( ),*node*$,*ident*$,*ifile*$)

Arguments:

lun

specifies the logical unit number of the logical link to execute. See the definition in Section 3.9.7.

* *status*

specifies completion status information on return from EXENFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file to execute. See the definition in Section 3.9.7.

ident

is an array/string containing explicit access control information. See the definition in Section 3.9.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be executed. See the definition in Section 3.9.7.

GETNFW
(Read a Single Record)

3.9.13 GETNFW — Read a Single Record

Use:

Call GETNFW to read a record from a file. The FORTRAN *inarray* or the COBOL or BASIC *instring* argument specifies the array/string in which to store the record. Each successive GETNFW call reads the record into the same array/string, overlaying any previous record. The previous record is no longer available in the user record storage area.

The optional *rac* argument specifies the record access mode. If you omit this argument, the default access mode is sequential file transfer by records. The records are read sequentially from the first record in the file.

If you include a *rac* argument specifying random access, you must include the *keyptr* argument to specify the record to read.

If an error occurs while a file is being read, the logical link is maintained. You must call CLSNFW to close the file.

Formats:

FORTRAN: CALL GETNFW (*lun*,*status*,*inbytes*,*inarray*,
[*seqno*],[*rac*],[*keyptr*],[*rop*])

COBOL: CALL "GETNFW" USING *lun*,*status*,*inchars*,
instring,[*seqno*],[*rac*],[*keyptr*],[*rop*].

BASIC: CALL GETNFW BY REF (*lun*%,*status*%( ),*inchars*%,*instring*$,
[*seqno*%],[*rac*%],[*keyptr*%( )],[*rop*%( )])

Arguments:

lun

specifies the logical unit number of the logical link created for reading your records. See the definition in Section 3.9.7. Use the same LUN you assigned in the OPRNFW, OPMNFW, or OPUNFW call.

* *status*

  specifies completion status information on return from GETNFW. The second word contains the byte count length of the record returned. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

  inbytes/inchars

  specifies the length in bytes/characters of *inarray/instring*. It is an integer variable or constant. The actual length of the record read is returned in the second status word.

* *inarray/instring*

  specifies the array/string that contains the record to read from the file. If the record size is larger than the integer you specified in *inbytes/inchars*, the balance of the record is lost.

* *seqno*

  specifies the sequence number for the record to read from the file. You must specify this integer variable for sequenced variable length records. If the record type is not sequenced variable length (or in RMS terms, variable with fixed control, VFC) the *seqno* argument is ignored. Be sure to specify the record type in the *ichar* argument of an open call.

  rac

  specifies the mode to use in accessing the file. If you omit the *rac* value, the default access mode is sequential file transfer by records. Sequential file transfer modes (3 and 5) cause any *rac* value in a subsequent GETNFW or PUTNFW calls to be ignored until you close the file. If the file is open for record access, the *rac* argument can be one of the following:

0	Sequential by record
1	Random by relative record number (RRN)
2	Random by record file address (RFA)
3	Sequential file transfer by records (default)

  If the file is open for block access, the *rac* argument can be one of the following:

4	Random blocks by virtual block number (VBN)
5	Sequential file transfer by blocks

keyptr

specifies the record. The length is assumed from the *rac* argument value.

RAC	Key
1	Two-word binary value of RRN Low-order word first
2	Three-word binary RFA Low-order word first
4	Two-word binary value of VBN Low-order word first

rop

specifies record processing options. Valid values are:

1	Position to EOF
4	Update if existing record

OPANFW, OPMNFW, OPRNFW, OPUNFW
(Open a File for Appending, Modifying, Reading, Updating Records)

3.9.14 OPANFW — Open a File for Appending Records

OPMNFW — Open a File for Modifying Records

OPRNFW — Open a File for Reading Records

OPUNFW — Open a File for Updating Records

Use:

Call one of the following subroutines to open an existing file:

Call OPANFW to open a sequential file for appending records.

Call OPMNFW to open and modify a sequential file.

Call OPRNFW to open a sequential file for reading records.

Call OPUNFW to open and update a sequential file.

For information on OPWNFW (Create and Open a File for Writing Records), refer to Section 3.9.18.

Formats:

FORTRAN: CALL {OPANFW / OPMNFW / OPRNFW / OPUNFW} (*lun,status,node,ident,ifile,ichar,len,* [*iblock*])

COBOL: CALL {"OPANFW" / "OPMNFW" / "OPRNFW" / "OPUNFW"} USING *lun,status,node,ident,ifile,ichar, len,*[*iblock*].

BASIC: CALL {OPANFW / OPMNFW / OPRNFW / OPUNFW} BY REF (*lun*%,*status*%( ),*node*$,*ident*$, *ifile*$,*ichar*$,*len*%,[*iblock*])

Arguments:

lun

specifies the logical unit number of the logical link created for the OPANFW, OPMNFW, OPRNFW, or OPUNFW call. Use the same LUN for any succeeding PUTNFW, PRGNFW, or CLSNFW call. See the definition in Section 3.9.7.

* *status*

specifies completion status information on return from OPANFW, OPMNFW, OPRNFW, or OPUNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file to open. See the definition in Section 3.9.7.

ident

is an array/string containing explicit access control information. See the definition in Section 3.9.7.

ifile

specifies an ASCIZ string containing the file specification for the file to open. See the definition in Section 3.9.7.

* *ichar*

is a 3-element array/string. If the values you specify differ from those stored in the file, the stored values are used. When the open call completes, the *ichar* array/string contains the stored values. Check these values to see how the file was actually opened. Make sure you specify the appropriate ASCII letter code as defined in the following three fields:

ichar(1) – Mode

Letter Code	Description
A	ASCII file
I	Binary image file

ichar(2) – Record Format

Letter Code	Description
U	Undefined format records
F	Fixed length records
V	Variable length records
S	Sequenced variable length records (VFC)
A	ASCII stream format

ichar(3) – Carriage Control

Letter Code	Description
F	FORTRAN carriage control
T	Terminal carriage control
N	No carriage control
P	Print file VFC

The following example displays one method for the *ichar* argument. In this example, *ichar* specifies the file to be opened as an ASCII file ('A'), with variable length records ('V'), and FORTRAN style carriage control ('F').

Example:

```
BYTE ICHAR (3)

DATA ICHAR/'A','V','F'/

ICHAR PIC XXX VALUE "AVF".   (COBOL)

ICHAR$="AVF"   (BASIC)
```

* *len*

is an integer variable that specifies record length. If the file has variable length records, enter the maximum record length. A null value (0) implies there is no maximum record length.

* *iblock*

is an integer variable that returns the number of blocks currently allocated to the file. The values are described as follows:

Entry	Description
$+n$	Number of noncontiguous blocks (where n = number of blocks)
$-n$	Number of contiguous blocks (where n = number of blocks)

PRGNFW
(Discard an Opened File)

3.9.15 PRGNFW — Discard an Opened File

Use:

Call PRGNFW to close a remote file because one or more errors occurred in the transfer. If the file was newly created by an OPWNFW, SPLNFW, or SUBNFW call, it is deleted. If the file existed previously and was just opened by an OPRNFW or OPANFW call, it is closed in its current state.

Formats:

FORTRAN: CALL PRGNFW (*lun,status*)

COBOL: CALL "PRGNFW" USING *lun,status*.

BASIC: CALL PRGNFW BY REF (*lun*%,*status*%( ))

Arguments:

lun

specifies the logical unit number of the logical link to close. See the definition in Section 3.9.7. Use the same LUN specified in the previous open call.

* *status*

specifies completion status information on return from PRGNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

PUTNFW
(Write a Single Record)

3.9.16 PUTNFW — Write a Single Record

Use:

Call PUTNFW to write a record to a file. PUTNFW writes the indicated number of bytes/characters from the array/string you specify in the *outarray/outstring* argument.

The optional *rac* argument specifies the record access mode. If you omit this argument, the default access mode is sequential file transfer by records. The records are written sequentially beginning at the first record position unless the file was opened with an OPANFW call. In that case they are written after the last record.

If a *rac* value specifies random access, the *keyptr* argument must specify the record position where the record is written.

If a PUTNFW call returns an error, you can close the output file with either a PRGNFW or CLSNFW call or continue the write (PUTNFW) operation.

Formats:

FORTRAN: CALL PUTNFW (*lun,status,outbytes,outarray,*
[seqno],[rac],[keyptr],[rop])

COBOL: CALL "PUTNFW" USING *lun,status,outchars,outstring,*
[seqno],[rac],[keyptr],[rop].

BASIC: CALL PUTNFW BY REF (*lun%,status%( ),outchars%,outstring$,*
[seqno%],[rac%],[keyptr%( )],[rop%( )])

Arguments:

lun

specifies the logical unit number of the logical link created for writing a single record. See the definition in Section 3.9.7. Use the same LUN you assigned in the OPANFW, OPMNFW, OPUNFW, SPLNFW, SUBNFW, or OPWNFW call.

* *status*

specifies completion status information on return from PUTNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

outbytes/outchars

specifies the number of bytes/characters to be written to the file from the *outarray/outstring* argument. This integer variable or constant must be equal to or less than the maximum record length you specified in the open call. If data overrun occurs, the remaining bytes are lost.

outarray/outstring

is the name of the array/string that contains the record to be written to the file.

seqno

specifies the sequence number of the record to be written. You must specify this integer variable or constant for sequenced variable length records. If the record type is not sequenced variable length (or in RMS terms, variable with fixed control, VFC), the *seqno* argument is ignored. Remember to specify the record type in the *ichar* argument of an open call.

rac

specifies the mode to use in accessing the file. If you omit the *rac* value, the default access mode is sequential file transfer by records. Sequential file transfer modes (3 and 5) cause any *rac* value in a subsequent GETNFW or PUTNFW calls to be ignored until you close the file. If the file is open for record access, the *rac* argument can be one of the following:

0	Sequential by record
1	Random by relative record number (RRN)
2	Random by record file address (RFA)
3	Sequential file transfer by records (default)

If the file is open for block access, the *rac* argument can be one of the following:

4	Random blocks by virtual block number (VBN)
5	Sequential file transfer by blocks

keyptr

specifies the record. The length is assumed from the *rac* argument value.

RAC	Key
1	Two-word binary value of RRN Low-order word first
2	Three-word binary RFA Low-order word first
4	Two-word binary value of VBN Low-order word first

rop

specifies record-processing options. Valid values are:

1	Position to EOF
4	Update if existing record

RENNFW
(Rename a File)

3.9.17 RENNFW — Rename a File

Use:

Call RENNFW to rename a remote file.

To return the new, fully-qualified file specification after the rename operation, use the set extended attributes (ATTNFW) call. Issue ATTNFW before issuing RENNFW, and use the *name* argument to specify a name buffer. On completion of the RENNFW call, the name buffer will contain the resulting file specification.

Formats:

FORTRAN: CALL RENNFW (*lun,status,node,ident,ofile,nfile*)

COBOL: CALL "RENNFW" USING *lun,status,node,ident,ofile,nfile.*

BASIC: CALL RENNFW BY REF (*lun*%,*status*%( ),*node*$,*ident*$, *ofile*$,*nfile*$)

Arguments:

lun

specifies the logical unit number of the logical link over which to rename a remote file. See the definition in Section 3.9.7.

* *status*

specifies completion status information on return from RENNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node on which to rename the file. See the definition in Section 3.9.7.

ident

is an array/string containing explicit access control information. See the definition in Section 3.9.7.

ofile

specifies an ASCIZ array/string containing the current name of the file to rename.

nfile

specifies an ASCIZ array/string containing the new file specification for the file to rename.

SPLNFW, SUBNFW, OPWNFW

3.9.18 SPLNFW — Create, Write, and Print a File

SUBNFW — Create, Write, and Execute a File

OPWNFW — Create and Open a File for Writing Records

Use:

Call one of the following subroutines to create a file:

Call SPLNFW	to create, write to, and print a new remote file at the remote node.
Call SUBNFW	to create, write to, and submit a new remote file to the remote batch/command file processor for execution. The file is deleted after execution. Successful completion of this call implies that the remote node handled the file properly, but not that the file ran or ran properly.
Call OPWNFW	to create and open a sequential file for writing records.

Formats:

FORTRAN: CALL { SUBNFW / SPLNFW / OPWNFW } (*lun,status,node,ident,ifile,ichar,len,* [*iblock*])

COBOL: CALL { "SUBNFW" / "SPLNFW" / "OPWNFW" } USING *lun,status,node,ident,ifile,*
ichar,len,[*iblock*].

BASIC: CALL { SUBNFW / SPLNFW / OPWNFW } BY REF (*lun*%,*status*%( ),*node*$,*ident*$,
ifile$,*ichar*$,*len*%,[*iblock*%])

Arguments:

lun

specifies the logical unit number of the logical link for the SPLNFW, SUBNFW, or OPWNFW call. Use the same LUN for any succeeding PUTNFW, PRGNFW, or CLSNFW call. See the definition in Section 3.9.7.

* *status*

specifies status completion information on return from SPLNFW, SUBNFW, or OPWNFW. See the definition for your language in Section 3.9.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file to open using SPLNFW, SUBNFW or OPWNFW. See the definition in Section 3.9.7.

ident

is an array/string containing explicit access control information. See the definition in Section 3.9.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be opened using SPLNFW, SUBNFW, or OPWNFW.

* *ichar*

is a 3-element array/string. If the values you specify differ from those stored in the file, the stored values are used. When the open call completes, the *ichar* array/string contains the stored values. Check these values to see how the file was actually opened. Make sure you specify the appropriate ASCII letter code as defined in the following three fields:

ichar(1) – Mode

Letter Code	Description
A	ASCII file
I	Binary image file

ichar(2) – Record Format

Letter Code	Description
U	Undefined format records
F	Fixed length records
V	Variable length records
S	Sequenced variable length records (VFC)
A	ASCII stream format

ichar(3) – Carriage Control

Letter Code	Description
F	FORTRAN carriage control
T	Terminal carriage control
N	No carriage control
P	Print file VFC

The *ichar* array/string specifies values for the new file.

* *len*

is an integer variable that specifies record length. If record lengths vary, enter the maximum record length. A null value (0) implies there is no maximum record length.

* *iblock*

is an integer variable that specifies the number of blocks to allocate for file creation. Enter one of the following values:

Entry	Description
0	Dynamic allocation
+ *n*	Number of noncontiguous blocks (where *n* = number of blocks)
-*n*	Number of contiguous blocks (where *n* = number of blocks)

When SPLNFW, SUBNFW, or OPWNFW completes, *iblock* specifies the number of blocks allocated (if you specified a + *n* or a -*n* argument), or 0 (if you specified dynamic allocation).

If the system cannot allocate the number of requested blocks, an error returns and frees the LUN. If you omit the *iblock* argument, the system allocates space dynamically.

3.9.19 FORTRAN Remote File Access Programming Examples

The following programs illustrate FORTRAN remote file access. The first example appends a local file to a remote file. The second example reads the contents of one remote file into another.

These examples are included in your tape or disk kit.

3.9.19.1 Append Example

The FTNAPP program appends the contents of a local ASCII file to the end of a remote ASCII file and then closes both files. If an error occurs, the program displays an error message.

```
C
C Copyright (C) 1983, 1985, 1986, 1987 by
C Digital Equipment Corporation, Maynard, Mass.
C
C
C This software is furnished under a license and may be used and  copied
C only  in  accordance  with  the  terms  of  such  license and with the
C inclusion of the above copyright notice.  This software or  any  other
C copies  thereof may not be provided or otherwise made available to any
C other person.  No title to and ownership of  the  software  is  hereby
C transferred.
C
C The information in this software is subject to change  without  notice
C and  should  not  be  construed  as  a commitment by Digital Equipment
C Corporation.
C
C Digital assumes no responsibility for the use or  reliability  of  its
C software on equipment which is not supplied by Digital.
C          .

C
C   FTNAPP.FTN -- Append a local ASCII file to a remote ASCII file
C
C   This program illustrates DECnet remote file access support for FORTRAN.
C
C   To task build, use the following command string:
C
C   FTNAPP,FTNAPP = FTNAPP
C   LB:[1,1]F4POTS/LB
C   LB:[1,1]NETFOR/LB
C   LB:[1,1]NETFOR/LB:NFAFSR
C   LB:[1,1]RMSLIB/LB           (if RMS is included)
C   /
C   UNITS=10
C   EXTSCT=$$FSR1:2700
C   ACTFIL=4
C   EXTTSK=1000                 (if RMS is included)
C   //
C
        BYTE            UID(40),PAS(40),ACC(40),NOD(7)
        BYTE            INPFIL(65),OUTFIL(65)
        BYTE            BUFFER(128),IDENT(120),ICHAR(3)
        INTEGER         NETLUN,INPLUN,OUTLUN,LNKNUM,MBXFLG
        INTEGER         UIDLEN,PASLEN,ACCLEN,NODLEN
        INTEGER         INPLEN,OUTLEN,ISTAT(2),MSTAT(3),IDENTL
        LOGICAL         EOF
        COMMON          IDENTL,IDENT
C
C   Initialize LUNs for the network, input file, and output file
C
        DATA            NETLUN,INPLUN,OUTLUN /1,2,3/
C
C   ASCII files, Variable length records and FORTRAN carriage control
C
```

(continued on next page)

```
      DATA ICHAR/'A','V','F'/
C
C   Build a user ID string in the IDENT array
C
      IDENTL = 0
      TYPE 100                                  ! Prompt for UID
      ACCEPT 130,UIDLEN,(UID(I), I=1,UIDLEN)    ! Read in a string
      CALL BLDID(UID,UIDLEN)                    ! Store UID into IDENT array
C
C   Build a password string in the IDENT array
C
      TYPE 110                                  ! Prompt for PAS
      ACCEPT 130,PASLEN,(PAS(I), I=1,PASLEN)    ! Read in a string
      CALL BLDID(PAS,PASLEN)                    ! Store PAS into IDENT array
C
C   Build an account number string in the IDENT array
C
      TYPE 120                                  ! Prompt for ACC
      ACCEPT 130,ACCLEN,(ACC(I), I=1,ACCLEN)    ! Read in a string
      CALL BLDID(ACC,ACCLEN)                    ! Store ACC into IDENT array
C
C   Build a remote node name string
C
      TYPE 140                                  ! Prompt for a node name
      ACCEPT 150,NODLEN,(NOD(I), I=1,NODLEN)    ! Read in a string
      NOD(NODLEN+1) = 0                         ! Terminate nodename string
C
C   Build a local input filename string
C
      TYPE 160                                  ! Prompt for input filename
      ACCEPT 180,INPLEN,(INPFIL(I), I=1,INPLEN)!Read in a string
      INPFIL(INPLEN+1)=0                        ! Terminate input file string
C
C   Build a remote output filename string
C
      TYPE 170                                  ! Prompt for output filename
      ACCEPT 180,OUTLEN,(OUTFIL(I), I=1,OUTLEN)!Read in a string
      OUTFIL(OUTLEN+1)=0                        ! Terminate output file string
C
C   Open access to the network - only one link, use long connect block
C
      LNKNUM = 1                                ! Allow only one link
      MBXBLF = 1                                ! Set long connect block flag
      EOF = .FALSE.                             ! Clear end-of-file flag

      CALL OPNNTW (NETLUN,ISTAT,MSTAT,LNKNUM,,MBXFLG) ! Access the network
      IF (ISTAT(1) .EQ. 1) GOTO 10                    ! If success, proceed
      TYPE *,'Cannot open network, status = ', ISTAT  ! Else, report error
      GOTO 90                                         !   and finish
C
C   Open local input and remote output files
C
10    OPEN (UNIT=INPLUN,NAME=INPFIL,TYPE='OLD',READONLY,ERR=50) ! Open input
      CALL OPANFW (OUTLUN,ISTAT,NOD,IDENT,OUTFIL,ICHAR,LENGTH)  ! Open output
      IF (ISTAT(1) .EQ. 1) GOTO 20                    ! If OK, proceed
```

```
        TYPE *,'Cannot open output file, status = ', ISTAT ! Else, report error
        GOTO 70                                           !   and finish
C
C   Main loop - read record from local file, write to remote file
C
20      READ (INPLUN,190,END=30,ERR=40) ICNT3,(BUFFER(I),I=1,ICNT3)
        CALL PUTNFW (OUTLUN,ISTAT,ICNT3,BUFFER) ! Put/write record to output
        IF (ISTAT(1) .EQ. 1) GOTO 20            ! If success, loop
        TYPE *,'Write error, status = ', ISTAT  ! Else, report error
        GOTO 60                                 !   and finish
C
C   Last read is complete. Print error message if not end of file.
C
30      EOF = .TRUE.                            ! Indicate normal completion
40      IF (EOF) GOTO 60                        ! No read error if end-of-file
        TYPE *,'Read error, status = ', ISTAT   ! Else, print error message
        GOTO 60                                 !   and finish
C
C   Process FORTRAN OPEN error
C       .
50      TYPE *,'Cannot open local input file'   ! Indicate that OPEN failed
        GOTO 80                           !   and finish
C
C   Finish - close files, deaccess the network, print status and exit
C
60      CALL CLSNFW (OUTLUN,ISTAT)              ! Close remote output file
70      CLOSE (UNIT=INPLUN)                     ! Close local input file
80      CALL CLSNTW (ISTAT)                     ! Deaccess the network
90      IF (EOF)       TYPE *, 'Successful completion'
        IF (.NOT. EOF) TYPE *, 'Error completion'
        STOP
C
C       Formats
C
100     FORMAT ('$User ID (39 char. max.): ')
110     FORMAT ('$Password (39 char. max.): ')
120     FORMAT ('$Account (39 char. max.): ')
130     FORMAT (Q,39A1)
140     FORMAT ('$Node (6 char. max.): ')
150     FORMAT (Q,6A1)
160     FORMAT ('$Input file (64 char. max.): ')
170     FORMAT ('$Output file (64 char. max.): ')
180     FORMAT (Q,64A1)
190     FORMAT (Q,128A1)

        END
C
C   BLDID (fld,fldlen) - Build an ASCIZ IDENT field
C
        SUBROUTINE      BLDID (FLD,LEN)
        BYTE            IDENT(30),FLD(80)
        INTEGER         IDENTL,LEN
        COMMON          IDENTL,IDENT
C
        DO 10, I=1,LEN
```

(continued on next page)

```
        IDENT(IDENTL+I) = FLD(I)
10      CONTINUE
        IDENTL = IDENTL+I
        IDENT(IDENTL) = 0
        RETURN

        END
```

3.9.19.2 Read/Write Example

The FTNRRW program reads the contents of one remote file into another remote file. When the program encounters an end-of-file character, the last record is written to the remote file and both files are closed. If a read or write error occurs, the program displays a message and exits.

```
C
C Copyright (C) 1983, 1985, 1986, 1987 by
C Digital Equipment Corporation, Maynard, Mass.
C
C
C This software is furnished under a license and may be used and  copied
C only  in  accordance  with  the  terms  of  such  license and with the
C inclusion of the above copyright notice.  This software or  any  other
C copies  thereof may not be provided or otherwise made available to any
C other person.  No title to and ownership of  the  software  is  hereby
C transferred.
C
C The information in this software is subject to change  without  notice
C and  should  not  be  construed  as  a commitment by Digital Equipment
C Corporation.
C
C Digital assumes no responsibility for the use or  reliability  of  its
C software on equipment which is not supplied by Digital.
C


C
C   FTNRRW.FTN - Read records from one remote file, write to another
C
C   This program illustrates DECnet remote file access support for FORTRAN.
C
C   To task build, use the following command string:
C
C   FTNRRW,FTNRRW = FTNRRW
C   LB:[1,1]F4POTS/LB
C   LB:[1,1]NETFOR/LB
C   LB:[1,1]NETFOR/LB:NFAFSR
C   LB:[1,1]RMSLIB/LB           (if RMS is included)
C   /
C   UNITS=10
C   EXTSCT=$$FSR1:10000
C   ACTFIL=4
C   EXTTSK=1000                 (if RMS is included)
C   //
C
C   Remote input file:  RNODE1"NFAR PRIV 123"::FTNRRW.INP
C   Remote output file: RNODE2"NFAR PRIV 123"::FTNRRW.OUT
C
        INTEGER         NETLUN,INPLUN,OUTLUN
        INTEGER         MBXFLG,LNKNUM
        INTEGER         ISTAT(2),RECLEN
        BYTE            IDINFO(14),ICHARS(3),RECBUF(512)
        LOGICAL         EOF
C
C   Specify LUNs for the network, input file, and output file
C
        DATA            NETLUN,INPLUN,OUTLUN /7,1,2/
C
C   Specify ASCIZ IDENT strings: user, password and account
```

(continued on next page)

```
C
      DATA IDINFO/'N','F','A','R',0,'P','R','I','V',0,'1','2','3',0/
C
C   Image mode, Variable length records and Terminal carriage control
C
      DATA ICHARS /'I','V','T'/
C
C   Initialize some flags
C
      MBXFLG = 1                              ! Set long connect block flag
      LNKNUM = 2                              ! Set number of links
      EOF = .FALSE.                           ! Initialize end-of-file flag
C
C   Open access to the network - allow two links, use long connect block
C
      CALL OPNNTW(NETLUN,ISTAT,,LNKNUM,,MBXFLG)
      IF (ISTAT(1) .EQ. 1) GOTO 10
      TYPE *, 'Cannot access network, status = ', ISTAT
      GOTO 80
C
C   Open remote input file
C
10    CALL OPRNFW(INPLUN,ISTAT,'RNODE1',IDINFO,
      1           '[NFAR]FTNRRW.INP',
      1           ICHARS,RECLEN)
      IF (ISTAT(1) .EQ. 1) GOTO 20
      TYPE *, 'Cannot open remote input file', ISTAT
      GOTO 70
C
C   Open remote output file
C
20    CALL OPWNFW(OUTLUN,ISTAT,'RNODE2',IDINFO,
      1           '[NFAR]FTNRRW.OUT',
      1           ICHARS,RECLEN)
      IF (ISTAT(1) .EQ. 1) GOTO 30
      TYPE *, 'Cannot open remote output file', ISTAT
      GOTO 60
C
C   Main loop - transfer records until end-of-file
C
C   Get a record from the input file.
C
30    CALL GETNFW(INPLUN,ISTAT,512,RECBUF)
      IF (ISTAT(1) .NE. 1) GOTO 40     ! If read error, check for EOF
      RECLEN = ISTAT(2)                ! Set number of bytes to write
C
C   Put the record to the output file
C
      CALL PUTNFW(OUTLUN,ISTAT,RECLEN,RECBUF)
      IF (ISTAT(1) .EQ. 1) GOTO 30     ! If write succeeded, loop
      TYPE *, 'Write error, status = ', ISTAT
      GOTO 50
C
C   The last read failed. Print error message if not an end-of-file.
C
40    IF (ISTAT(2) .EQ. '050047'O) EOF = .TRUE.
      IF (.NOT. EOF) TYPE *,'Read error, status = ',ISTAT
C
C   Finish - close files, deaccess network, print status and exit
C
50    CALL CLSNFW(OUTLUN,ISTAT)        ! Close output file
60    CALL CLSNFW(INPLUN,ISTAT)        ! Close input file
70    CALL CLSNTW(NETLUN)              ! Deaccess the network
80    IF (EOF)       TYPE *, 'Successful completion'
      IF (.NOT. EOF) TYPE *, 'Execution failure'
      STOP
      END
```

3.9.20 COBOL Remote File Access Programming Examples

The following programs illustrate COBOL remote file access. They are included in your tape or disk kit.

3.9.20.1 Append Example

The program COBAPP appends the contents of a local ASCII file to the end of a remote ASCII file and then closes both files. If an error occurs, the program displays an error message.

```
*
* Copyright (C) 1983, 1985, 1986, 1987 by
* Digital Equipment Corporation, Maynard, Mass.
*
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by Digital Equipment
* Corporation.
*
* Digital assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by Digital.
*


IDENTIFICATION DIVISION.
PROGRAM-ID. COBAPP.

*****************************************************************
*                                                               *
*  This program appends the contents of a local ASCII file      *
*  to a remote ASCII file and then closes both files.           *
*                                                               *
*  To task build, use the following command string:             *
*                                                               *
*  COBAPP,COBAPP =-                                             *
*  COBAPP,[1,1]NETFOR/LB,C81LIB/LB,RMSLIB/LB,NETFOR/LB:NFAFSR   *
*  /                                                            *
*  UNITS=10                                                     *
*  EXTSCT=$$FSR1:2700                                           *
*  ACTFIL=4                                                     *
*  EXTTSK=1000                  (if RMS is included)            *
*  //                                                           *
*                                                               *
*****************************************************************

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT LOCAL-FILE ASSIGN TO "DB0:".
DATA DIVISION.
FILE SECTION.

FD      LOCAL-FILE
        LABEL RECORDS ARE STANDARD
        VALUE OF ID IS LOCAL.
```

```
01  LOCAL-REC                  PIC X(80).

WORKING-STORAGE SECTION.
01  MSGS.

    03  MSG1.
        05  FILLER             PIC X(36)     VALUE " MAIL BOX CREAT
-                               "ION ERROR, IOST(1) = ".
        05  MSG1-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG1-STAT2         PIC -99999.
    03  MSG2.
        05  FILLER             PIC X(38)     VALUE " CAN NOT OPEN R
-                               "EMOTE FILE.  IOST(1) = ".
        05  MSG2-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG2-STAT2         PIC -99999.
    03  MSG3.
        05  FILLER             PIC X(42)     VALUE " WRITE ERROR FR
-                               "OM REMOTE FILE.  IOST(1) = ".
        05  MSG3-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG3-STAT2         PIC -99999.
    03  MSG4.
        05  FILLER             PIC X(39)     VALUE " CAN NOT CLOSE
-                               "REMOTE FILE.  IOST(1) = ".
        05  MSG4-STAT1         PIC -99999.
        05 .FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG4-STAT2         PIC -99999.
    03  MSG5.
        05  FILLER             PIC X(35)     VALUE " CAN NOT CLOSE
-                               "NETWORK.  IOST(1) = ".
        05  MSG5-STAT1         PIC -99999.
        05  FILLER             PIC X(11)      VALUE "IOST(2) = ".
        05  MSG5-STAT2         PIC -99999.
01  ARRAYS.
    03  IOST.
        05  IOSTAT OCCURS 2 TIMES PIC S9999 USAGE COMP.
    03  MSTAT.
        05  MSTATS OCCURS 3 TIMES PIC S9999 USAGE COMP.
01  STORE-STUFF.
    03  LOCAL                  PIC X(26).
    03  IDENT                  PIC X(30).
    03  USERID                 PIC X(12).
    03  PASSWD                 PIC X(6).
    03  ACCNT                  PIC X(9).
    03  REMOTE-FILE            PIC X(30).
    03  FILLER                 PIC X.
    03  OPNLUN                 PIC 9         COMP VALUE 4.
    03  COUNT1                 PIC 9         COMP VALUE 1.
    03  APPLUN                 PIC 9         COMP VALUE 3.
    03  LENGTH1                PIC S9999     USAGE COMP.
    03  BLOCK1                 PIC S9999     USAGE COMP.
    03  REC-LENGTH             PIC S99       COMP VALUE 80.
    03  NODE-NAME              PIC X(7).
    03  TEMP-NODE              PIC X(6).
    03  TEMP-REMOTE            PIC X(29).
    03  ICHAR                  PIC X(3)      VALUE "AVF".
```

(continued on next page)

```
01  NULL1                      PIC 9          COMP VALUE 0.
01  NULLS REDEFINES NULL1.
    03 NUL OCCURS 2 TIMES   PIC X(1).
PROCEDURE DIVISION.


****************************************************************
*                                                              *
*       Get accounting information for remote node from        *
*       terminal and form ASCIZ string with this information   *
*       for OPRNFW and OPWNFW.                                 *
*                                                              *
****************************************************************

A100-START.
        DISPLAY " INPUT USER ID: ".
        ACCEPT USERID.
        DISPLAY " INPUT PASSWORD: ".
        ACCEPT PASSWD.
        DISPLAY " INPUT ACCOUNT NUMBER: ".
        ACCEPT ACCNT.
        STRING USERID
                NUL(1)
                PASSWD
                NUL(1)
                ACCNT
                NUL(1) DELIMITED BY SIZE
                INTO IDENT.

****************************************************************
*                                                              *
*       Get remote node name and form ASCIZ string.            *
*                                                              *
****************************************************************

        DISPLAY " INPUT REMOTE NODE NAME: ".
        ACCEPT TEMP-NODE.
        STRING TEMP-NODE
                NUL(1) DELIMITED BY SIZE
                INTO NODE-NAME.

****************************************************************
*                                                              *
*       Get remote file name and form ASCIZ string.            *
*                                                              *
****************************************************************

        DISPLAY " ENTER FILE SPEC. OF REMOTE FILE FOR APPEND".
        ACCEPT TEMP-REMOTE.
        STRING TEMP-REMOTE
                NUL(1) DELIMITED BY SIZE
                INTO REMOTE-FILE.
```

```
*****************************************************************
*                                                               *
*       Get local file name.                                    *
*                                                               *
*****************************************************************

        DISPLAY " ENTER FILE SPEC. OF LOCAL FILE TO BE APPENDED".
        ACCEPT LOCAL.

*****************************************************************
*                                                               *
*       Access the network.  If the call completes              *
*       unsuccessfully, write an error message and exit.        *
*                                                               *
*****************************************************************

        CALL "OPNNTW" USING

                OPNLUN
                IOST
                MSTAT
                COUNT1.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG1-STAT1
                MOVE IOSTAT (2) TO MSG1-STAT2
                DISPLAY MSG1
                GO E100-END.

*****************************************************************
*                                                               *
*       Open the local file.  Open the remote file for          *
*       append.  If unable to open the remote file, write       *
*       an error message and deaccess the network.              *
*                                                               *
*****************************************************************

        OPEN INPUT LOCAL-FILE.
        CALL "OPANFW" USING
                APPLUN
                IOST
                NODE-NAME
                IDENT
                REMOTE-FILE
                ICHAR
                LENGTH1
                BLOCK1.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG2-STAT1
                MOVE IOSTAT (2) TO MSG2-STAT2
                DISPLAY MSG2
                GO D100-CLOSE.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       Read a record from the local file and append it to      *
*       the remote file until the end-of-file is encountered    *
*       in the local file.  If an error occurs while writing    *
*       to the remote file, print an error message and exit.    *
*                                                               *
*****************************************************************


B100-READ.
        MOVE SPACES TO LOCAL-REC.
        READ LOCAL-FILE RECORD
                AT END GO C100-EOF.
        CALL "PUTNFW" USING
                APPLUN
                IOST
                REC-LENGTH
                LOCAL-REC.
        IF IOSTAT (1) = 1 GO B100-READ.
        MOVE IOSTAT (1) TO MSG3-STAT1.
        MOVE IOSTAT (2) TO MSG3-STAT2.
        DISPLAY MSG3.
        GO E100-END.



*****************************************************************
*                                                               *
*       When the end-of-file is encountered in the local        *
*       file, close the local and remote files.  If unable      *
*       to close the remote file, print an error message        *
*       and exit.                                               *
*                                                               *
*****************************************************************

C100-EOF.
        CLOSE LOCAL-FILE.
        CALL "CLSNFW" USING
                APPLUN
                IOST.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG4-STAT1
                MOVE IOSTAT (2) TO MSG4-STAT2
                DISPLAY MSG4
                GO E100-END.
```

```
*****************************************************************
*                                                               *
*       Deaccess the network.  Display an error message         *
*       if the call does not complete successfully.             *
*                                                               *
*****************************************************************

D100-CLOSE.
        CALL "CLSNTW" USING
                IOST.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG5-STAT1
                MOVE IOSTAT (2) TO MSG5-STAT2
                DISPLAY MSG5
                GO E100-END.
        DISPLAY "APPEND COMPLETE. END COBAPP PROGRAM EXECUTION".
E100-END.
        STOP RUN.
```

3.9.20.2 Read/Write Example

The program COBRRW reads the contents of one remote file into another remote file. When an end-of-file character is encountered, the last record is written to the remote file and both files are closed.

```
*
* Copyright (C) 1983, 1985, 1986, 1987 by
* Digital Equipment Corporation, Maynard, Mass.
*
*
* This software is furnished under a license and may be used and  copied
* only  in  accordance  with  the  terms  of  such  license and with the
* inclusion of the above copyright notice.  This software or  any  other
* copies  thereof may not be provided or otherwise made available to any
* other person.  No title to and ownership of  the  software  is  hereby
* transferred.
*
* The information in this software is subject to change  without  notice
* and  should  not  be  construed  as  a commitment by Digital Equipment
* Corporation.
*
* Digital assumes no responsibility for the use or  reliability  of  its
* software on equipment which is not supplied by Digital.
*


IDENTIFICATION DIVISION.
PROGRAM-ID. COBRRW.

*****************************************************************
*                                                               *
*  This program reads the contents of a remote file into        *
*  another remote file.  The program reads and writes records   *
*  until it encounters an end-of-file, at which time it writes  *
*  the last record to the remote file and closes both files.    *
*                                                               *
*  To task build, use the following command string:             *
*                                                               *
*  COBRRW,COBRRW =-                                             *
*  COBRRW,[1,1]NETFOR/LB,C81LIB/LB,RMSLIB/LB,NETFOR/LB:NFAFSR   *
*  /                                                            *
*  UNITS=10                                                     *
*  EXTSCT=$$FSR1:10000                                          *
*  ACTFIL=4                                                     *
*  EXTTSK=1000                  (if RMS is included)            *
*  //                                                           *
*                                                               *
*****************************************************************

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT DUMMY-FILE ASSIGN TO "COBRRW.DUM".

DATA DIVISION.
```

```
FILE SECTION.
FD      DUMMY-FILE
        LABEL RECORD STANDARD.
01      DUMMY-FILE-REC.
  02    FILLER                PIC X(132).
WORKING-STORAGE SECTION.
01  MSGS.
    03  MSG1.
        05  FILLER             PIC X(34)     VALUE " CAN NOT OPEN N
-                               "ETWORK.  IOST(1) = ".
        05  MSG1-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG1-STAT2         PIC -99999.
    03  MSG2.
        05  FILLER             PIC X(44)     VALUE " CAN NOT OPEN R
-                               "EMOTE INPUT FILE.  IOST(1) = ".
        05  MSG2-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05 MSG2-STAT2          PIC -99999.
    03  MSG3.
        05  FILLER             PIC X(45)     VALUE " CAN NOT OPEN R
-                               "EMOTE INPUT FILE.  IOST(1) = ".
        05  MSG3-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG3-STAT2         PIC -99999.
    03  MSG4.
        05  FILLER             PIC X(24)     VALUE " READ ERROR.  I
-                               "OST(1) = ".
        05  MSG4-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG4-STAT2         PIC -99999.
    03  MSG5.
        05  FILLER             PIC X(25)     VALUE " WRITE ERROR.
-                               "IOST(1) = ".
        05  MSG5-STAT1         PIC -99999.
        05  FILLER             PIC X(11)     VALUE " IOST(2) = ".
        05  MSG5-STAT2         PIC -99999.
01  ARRAYS.
    03  IOST.
        05  IOSTAT OCCURS 2 TIMES PIC S9999 USAGE COMP.
    03  MSTAT.
        05  MSTATS OCCURS 3 TIMES PIC S9999 USAGE COMP.
01  STORE-STUFF.
    03  OPNLUN                 PIC 9         COMP VALUE 2.
    03  COUNT1                 PIC 9         COMP VALUE 2.
    03  LENGTH1                PIC S9999     USAGE COMP.
    03  BLOCK1                 PIC S9999     USAGE COMP.
    03  INPLUN                 PIC 9         COMP VALUE 3.
    03  OUTLUN                 PIC 9         COMP VALUE 4.
    03  I                      PIC 999       USAGE COMP.
    03  IARRAY-SIZE            PIC 99        COMP VALUE 80.
    03  EOF                    PIC 99999     COMP VALUE 20519.
    03  EOFFG                  PIC S        USAGE COMP.
    03  TTRUE                  PIC S        COMP VALUE -1.
    03  FFALSE                 PIC S        COMP VALUE 0.
```

(continued on next page)

```
    03  IDENT                PIC X(30).
    03  USERID               PIC X(12).
    03  PASSWD               PIC X(6).
    03  ACCNT                PIC X(9).
    03  TEMP-NODE            PIC X(6).
    03  NODE-NAME            PIC X(7).
    03  TEMP-INPUT           PIC X(29).
    03  REMOTE-INPUT         PIC X(30).
    03  TEMP-OUTPUT          PIC X(29).
    03  REMOTE-OUTPUT        PIC X(30).
    03  ICHAR                PIC X(3)      VALUE "AVF".
    03  IARRAY               PIC X(80).
01  NULL1                    PIC 9         COMP VALUE 0.
01  NULLS REDEFINES NULL1.
    03  NUL OCCURS 2 TIMES  PIC X(1).
PROCEDURE DIVISION.

*****************************************************************
*                                                               *
*       Get.accounting information for remote node and          *
*       form ASCIZ string for DECnet remote file access         *
*       subroutines.                                            *
*                                                               *
*****************************************************************

A100-START.
        DISPLAY "INPUT USER ID:".
        ACCEPT USERID.
        DISPLAY " INPUT PASSWORD:".
        ACCEPT PASSWD.
        DISPLAY " INPUT ACCOUNT NUMBER:".
        ACCEPT ACCNT.
        STRING USERID
                NUL(1)
                PASSWD
                NUL(1)
                ACCNT
                NUL(1) DELIMITED BY SIZE
                INTO IDENT.

*****************************************************************
*                                                               *
*       Get remote node name and form ASCIZ string.             *
*                                                               *
*****************************************************************

        DISPLAY " ENTER REMOTE NODE NAME:".
        ACCEPT TEMP-NODE.
        STRING TEMP-NODE
                NUL(1) DELIMITED BY SIZE
                INTO NODE-NAME.

*****************************************************************
*                                                               *
*       Get remote input and output file names and form         *
*       ASCIZ string for each file.                             *
*                                                               *
*****************************************************************
```

```
        DISPLAY " INPUT FILE SPEC. FOR INPUT FILE:".
        ACCEPT TEMP-INPUT.
        STRING TEMP-INPUT
                NUL(1) DELIMITED BY SIZE
                INTO REMOTE-INPUT.
        DISPLAY " INPUT FILE SPEC. FOR OUTPUT FILE:".
        ACCEPT TEMP-OUTPUT.
        STRING TEMP-OUTPUT
                NUL(1) DELIMITED BY SIZE
                INTO REMOTE-OUTPUT.

*****************************************************************
*                                                               *
*       Access the network.  If the call does not complete      *
*       successfully, display an error message and exit.        *
*                                                               *
*****************************************************************

        CALL "OPNNTW" USING
                OPNLUN
                IOST
                MSTAT
                COUNT1.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG1-STAT1
                MOVE IOSTAT (2) TO MSG1-STAT2
                DISPLAY MSG1
                GO E100-END.

*****************************************************************
*                                                               *
*       Open remote file for input.  If there is an open        *
*       error, print an error message and exit.                 *
*                                                               *
*****************************************************************

        CALL "OPRNFW" USING
                INPLUN
                IOST
                NODE-NAME
                IDENT
                REMOTE-INPUT
                ICHAR
                LENGTH1
                BLOCK1.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG2-STAT1
```

(continued on next page)

```
                MOVE IOSTAT (2) TO MSG2-STAT2
                DISPLAY MSG2
                GO E100-END.

*****************************************************************
*                                                               *
*       Open remote file for output.  If there is an open       *
*       error, display an error message and exit.               *
*                                                               *
*****************************************************************

        CALL "OPWNFW" USING
                OUTLUN
                IOST
                NODE-NAME
                IDENT
                REMOTE-OUTPUT
                ICHAR
                ICHAR
                LENGTH1
                BLOCK1.
        IF IOSTAT (1) = 1
                NEXT SENTENCE
        ELSE
                MOVE IOSTAT (1) TO MSG3-STAT1
                MOVE IOSTAT (2) TO MSG3-STAT2
                DISPLAY MSG3
                GO D100-CLOSE.

*****************************************************************
*                                                               *
*       Transfer records between the remote files.  When        *
*       the end-of-file is encountered, exit from the loop.     *
*       Exit from the loop if a read or write error occurs      *
*       and branch to the appropriate routine to display an     *
*       error message.                                          *
*                                                               *
*****************************************************************

        PERFORM LOOP VARYING I FROM 1 BY 1 UNTIL I = 100.
LOOP.
        CALL "GETNFW" USING
                INPLUN
                IOST
                IARRAY-SIZE
                IARRAY.
        IF IOSTAT (1) NOT = 1 AND IOSTAT (2) NOT = EOF
                GO B100-READERR.
        MOVE IOSTAT (2) TO LENGTH1.
        IF IOSTAT (1) NOT = 1 AND IOSTAT (2) = EOF
                MOVE TTRUE TO EOFFG
        ELSE
                MOVE FFALSE TO EOFFG.
        IF EOFFG = TTRUE GO D100-CLOSE.
        CALL "PUTNFW" USING
```

```
                OUTLUN
                IOST
                LENGTH1
                IARRAY
        IF IOSTAT (1) NOT = 1 AND IOSTAT (2) NOT = EOF
                GO C100-WRITERR.

*****************************************************************
*                                                               *
*       A read error occurred during file transfer.  Print      *
*       an error message and exit.                              *
*                                                               *
*****************************************************************

B100-READERR.
        MOVE IOSTAT (1) TO MSG4-STAT1.
        MOVE IOSTAT (2) TO MSG4-STAT2.
        DISPLAY MSG4.
        GO D100-CLOSE.
           .
*****************************************************************
*                                                               *
*       A write error occurred during file transfer.  Print     *
*       an error message and exit.                              *
*                                                               *
*****************************************************************

C100-WRITERR.
        MOVE IOSTAT (1) TO MSG5-STAT1.
        MOVE IOSTAT (2) TO MSG5-STAT2.
        DISPLAY MSG5.

*****************************************************************
*                                                               *
*       Close both remote files.                                *
*                                                               *
*****************************************************************

D100-CLOSE.
        PERFORM LOOP1 VARYING I FROM INPLUN BY 1 UNTIL I = OUTLUN.
LOOP1.
        CALL "CLSNFW" USING
                I
                IOST.

*****************************************************************
*                                                               *
*       If an error occurred before encountering the            *
*       end-of-file, exit.  Otherwise the transfer was          *
*       successful, so display a success message and            *
*       exit.                                                   *
*                                                               *
*****************************************************************

END-LOOP1.

        IF EOFFG NOT = TTRUE
                GO E100-END
        ELSE
                DISPLAY "END OF FILE REACHED.  FILES CLOSED.".
E100-END.
        STOP RUN.
```

(continued on next page)

3.9.21 BASIC–PLUS–2 Remote File Access Programming Examples

The following programs illustrate BASIC–PLUS–2 remote file access. The first example appends a local file to a remote file. The second example reads the contents of one remote file into another.

These programs are included in your tape or disk kit.

3.9.21.1 Append Example

The BASAPP program appends the contents of a local ASCII file to the end of a remote ASCII file and then closes both files. If an error occurs, the program displays an error message. In the following example, the user ID, the password, and the account number are entered from the terminal.

```
!
! Copyright (C) 1983, 1985, 1986, 1987 by
! Digital Equipment Corporation, Maynard, Mass.
!
!
! This software is furnished under a license and may be used and  copied
! only  in  accordance  with  the  terms  of  such  license and with the
! inclusion of the above copyright notice.  This software or  any  other
! copies  thereof may not be provided or otherwise made available to any
! other person.  No title to and ownership of  the  software  is  hereby
! transferred.
!
! The information in this software is subject to change  without  notice
! and  should  not  be  construed  as  a commitment by Digital Equipment
! Corporation.
!
! Digital assumes no responsibility for the use or  reliability  of  its
! software on equipment which is not supplied by Digital.
!


10      !!!                                                         !!! &
        !!!    BASAPP.B2S - Append local file to remote file        !!! &
        !!!                                                         !!! &
        !!!    To task build, edit the task build command file      !!! &
        !!!    file and the ODL file created by the build.          !!! &
        !!!                                                         !!! &
        !!!    1) Add the lines                                     !!! &
        !!!             ACTFIL=4                                    !!! &
        !!!             EXTSCT=$$FSR1:2700                          !!! &
        !!!    to the task build command file.                      !!! &
        !!!                                                         !!! &
        !!!    2) Append                                            !!! &
        !!!             -NETLIB-NETLB2                              !!! &
        !!!    to the USER: line of the ODL file.                   !!! &
        !!!                                                         !!! &
        !!!    3) Add the lines                                     !!! &
        !!!             NETLIB: .FCTR LB:[1,1]NETFOR/LB             !!! &
        !!!             NETLB2: .FCTR LB:[1,1]NETFOR/LB:NFAFSR      !!! &
        !!!    to the ODL file.                                     !!! &

        ON ERROR GO TO 200              ! Error handler

20      !!! Define array constants !!!                                  &
 \      DIM ISTAT%(1%),JSTAT%(1%),KSTAT%(1%),LSTAT%(1%),MSTAT%(2%)     &
 \      DIM NSTAT%(1%)                  ! Define array elements         &
 \      NULL$ = STRING$(1%,0%)          ! Define null char for ASCIZ

30      !!! Define constants !!!                                        &
        OPNLUN% = 2%                    ! Network open LUN              &
 \      MBXFLG% = 1%                    ! Long connect block flag       &
 \      APPLUN% = 1%                    ! File LUN                      &
 \      COUNT% = 1%                     ! Max # of logical links        &
```

(continued on next page)

```
\       FLAG% = 0%                      ! End of file flag              &
\       ICHAR$ = "AVF"                  ! Mode, type, carriage control

40      INPUT "Remote node name (6 char. max.)";NODNAM$                 &
\       IF LEN(NODNAM$)>6% THEN PRINT                                   &
          "Node name too long, please re-enter"                         &
\           PRINT \ GO TO 40
50      NODNAM$ = NODNAM$+NULL$         ! Create ASCIZ string for OPANFW

60      INPUT "Remote output file (64 char. max.)";OFIL$                &
\       IF LEN(OFIL$)>64% THEN PRINT                                    &
          "Remote output filename too long, please re-enter"            &
\           PRINT \ GOTO 60
70      OFIL$ = OFIL$+NULL$             ! Create ASCIZ string for OPANFW

80      INPUT "Local input file (64 char. max)";IFIL$                   &
\       IF LEN(IFIL$)>64% THEN PRINT                                    &
          "Local input filename too long, please re-enter"              &
\           PRINT \ GOTO 80
            .
90      INPUT "User ID (39 char. max.)";UID$    ! Get user ID           &
\       IF LEN(UID$)>39% THEN PRINT                                     &
          "User ID too long, please re-enter"                           &
\           PRINT \ GOTO 90
100     INPUT "Password (39 char. max.)";PAS$   ! Get password          &
\       IF LEN(PAS$)>39% THEN PRINT                                     &
          "Password too long, please re-enter"                          &
\           PRINT \ GOTO 100
110     INPUT "Account (39 char. max.)";ACC$    ! Get account number    &
\       IF LEN(ACC$)>39% THEN PRINT                                     &
          "Account number too long, please re-enter"                    &
\           PRINT \ GOTO 110

120     !!! Create ASCIZ string for IDENT in OPANFW             !!!     &
\       IDENT$ = UID$+NULL$+PAS$+NULL$+ACC$+NULL$

130     !!! Open access to network - 1 link, long connect block !!!     &
\       CALL OPNNTW BY REF (OPNLUN%,LSTAT%(),MSTAT%(),COUNT%            &
                            ,,MBXFLG%)                                  &
\       IF LSTAT%(0%)=1% THEN 140       ! If OPNNTW succeeded, proceed
        ELSE PRINT "Cannot access network"      ! Else, print message   &
\            PRINT "Status = ";LSTAT%(0%);",";LSTAT%(1%)                &
\               GO TO 220                       !   status and exit

140     !!! Open local file                                     !!!     &
\       OPEN IFIL$ FOR INPUT AS FILE #4

150     !!! Open remote file for append                         !!!     &
\       CALL OPANFW BY REF(APPLUN%,ISTAT%(),NODNAM$,IDENT$,OFIL$,       &
                           ICHAR$,LENGTH%,IBLOCK%)                      &
\       IF ISTAT%(0%)=1% THEN 160       ! If successful, proceed        &
        ELSE PRINT "Cannot open remote file." ! Else, print message,    &
\         PRINT "STATUS = ";ISTAT%(0%);",";ISTAT%(1%) ! status          &
\           GO TO 180                                 !   and exit
```

```
160     !!!     Read records from local file and write them to  !!!     &
 \      !!!       remote file.                                  !!!     &
 \      FLAG% = 1%                      ! Set flag for eof check        &
 \      INPUT   #4,TEMP$                ! Read from local file          &
 \      CALL PUTNFW BY REF(APPLUN%,JSTAT%(),LEN(TEMP$),TEMP$)           &
                                        ! Append to remote file         &
 \      IF JSTAT%(0%)=1% THEN 160       ! If successful, loop           &
        ELSE PRINT "Write error from remote file."   ! Else, print msg, &
 \        PRINT "Status = ";JSTAT%(0%);",";JSTAT%(1%)!   status         &
 \          GO TO 220                                 !     and exit

170     !!!  EOF found -- close both files and network !!!              &
 \      FLAG% = 0%                      ! Clear end of file flag        &
 \      CLOSE #4                        ! Close local file              &
 \      CALL CLSNFW BY REF(APPLUN%,KSTAT%()) ! Close remote file        &
 \      IF KSTAT%(0%)=1% THEN 180       ! If success, deaccess network  &
        ELSE PRINT " Cannot close remote file." ! If close error,       &
 \        PRINT "Status = ";KSTAT%(0%);",";KSTAT%(1%)                   &
 \          GO TO 220           ! Print message, status and exit
          .
180     CALL CLSNT BY REF(NSTAT%())     ! Deaccess network              &
 \      IF NSTAT%(0%)=1% THEN 190       ! If success, append complete   &
        ELSE PRINT "Cannot close network." ! If error, print            &
 \        PRINT "Status = ";NSTAT%(0%);",";NSTAT$(1%)                   &
 \          GO TO 220           ! message, status and exit

190     PRINT "Append complete. End program execution"                  &
 \      GO TO 220

200     IF ERR <> 11 THEN 210           ! If not EOF, print error       &
        ELSE IF FLAG%=0% THEN 210       ! If EOF and EOF flag not set,  &
                                        !   print error                 &
        ELSE RESUME 170                 ! EOF so close both files

210     PRINT "Error ";ERR;" at line ";ERL ! Print error and line number

220     END
```

3.9.21.2 Read/Write Example

The BASRRW program reads the contents of one remote file into another remote file. When the program encounters an end-of-file character, the last record is written to the remote file and both files are closed.

```
!
! Copyright (C) 1983, 1985, 1986, 1987 by
! Digital Equipment Corporation, Maynard, Mass.
!
!
! This software is furnished under a license and may be used and  copied
! only  in  accordance  with  the  terms  of  such  license and with the
! inclusion of the above copyright notice.  This software or  any  other
! copies  thereof may not be provided or otherwise made available to any
! other person.  No title to and ownership of  the  software  is  hereby
! transferred.
!
! The information in this software is subject to change  without  notice
! and  should  not  be  construed  as  a commitment by Digital Equipment
! Corporation.
!
! Digital assumes no responsibility for the use or  reliability  of  its
! software on equipment which is not supplied by Digital.
!


10      !!!                                                         !!! &
        !!!    BASRRW.B2S - Read records from one remote file       !!! &
        !!!                   and write them to another             !!! &
        !!!                                                         !!! &
        !!!    To task build, edit the task build command file      !!! &
        !!!    and the ODL file created by the build.               !!! &
        !!!                                                         !!! &
        !!!     1) Add the lines                                    !!! &
        !!!           ACTFIL=4                                      !!! &
        !!!           EXTSCT=$$FSR1:10000                           !!! &
        !!!        to the task build command file.                  !!! &
        !!!                                                         !!! &
        !!!     2) Append                                           !!! &
        !!!           -NETLIB-NETLB2                                !!! &
        !!!        to the USER: line of the ODL file.               !!! &
        !!!                                                         !!! &
        !!!     3) Add the lines                                    !!! &
        !!!           NETLIB: .FCTR LB:[1,1]NETFOR/LB               !!! &
        !!!           NETLB2: .FCTR LB:[1,1]NETFOR/LB:NFAFSR        !!! &
        !!!        to the ODL file.                                 !!! &
        !!!                                                         !!!

20      !!! Define array constants !!!                                 &
        DIM     IARRAY%(255%),ISTAT%(1%),MSTAT%(2%)                    &
                                        ! Define maximum string lengths &
 \      NULL$ = STRING$(1%,0%)          ! Define maximum string lengths &

30      !!! Define constants !!!                                       &
        ICHARS$ = "AVF"                 ! Mode, type, carriage control  &
 \      OPNLUN% = 7%                    ! Network OPEN LUN              &
 \      COUNT% = 2%                     ! Max. # of active logical links&
```

```
\       MBXFLG% = 1                     ! Long connect block flag       &
\       INPLUN% = 1%                    ! Input file LUN                &
\       OUTLUN% = 2%                    ! Output file LUN               &
\       EOF% = 20519%                   ! End of file status return     &
\       FALSE% = 0%                     ! Flag indicating FALSE         &
\       TRUE% = -1%                     ! Flag indicating TRUE

40      INPUT "Remote node name (6 char. max.)";NODNAM$                 &
\       IF LEN(NODNAM$)>6% THEN PRINT                                   &
          "Node name too long, please re-enter"                         &
\           PRINT \ GO TO 40                                            &
        ELSE NODNAM$ = NODNAM$+NULL$            ! Form ASCIZ nodename

50      INPUT "User ID (39 char. max.)";UID$            ! Get user ID   &
\       IF LEN(UID$)>39% THEN PRINT                                     &
          "User ID too long, please re-enter"                           &
\           PRINT \ GOTO 50
60      INPUT "Password (39 char. max.)";PAS$           ! Get password  &
\       IF LEN(PAS$)>39% THEN PRINT                                     &
          "Password too long, please re-enter"                          &
\           PRINT \ GOTO 60
70      INPUT "Account number (39 char. max.)";ACC$     ! Get account   &
\       IF LEN(PAS$)>39% THEN PRINT                                     &
          "Account number too long, please re-enter"                    &
\           PRINT \ GOTO 70

80      !!! Form ASCIZ IDENT string for remote file opens       !!!     &
\       IDENT$ = UID$+NULL$+PAS$+NULL$+ACC$+NULL$

90      INPUT "Input file (64 char. max.)";IFIL$        ! Get inp file  &
\       IF LEN(IFIL$)>64% THEN PRINT                                    &
          "Input filename too long, please re-enter"                    &
\           PRINT \ GOTO 90                                             &
        ELSE IFIL$=IFIL$+NULL$                  ! Form ASCIZ filename

100     INPUT "Output file (64 char. max.)";OFIL$       ! Get out file  &
\       IF LEN(OFIL$)>64% THEN PRINT                                    &
          "Output filename too long, please re-enter"                   &
\           PRINT \ GOTO 100                                            &
        ELSE OFIL$=OFIL$+NULL$          ! Form ASCIZ filename

110     !!! Open access to network - 2 links, long connect block!!!     &
        CALL OPNNTW BY REF(OPNLUN%,ISTAT%(),MSTAT%(),CCUNT%             &
                           ,,MBXFLG)                                    &
\       LOC1 = 1                        ! Origin of CALL for subroutine &
\       GOSUB 250                       ! Check status

120     !!! Open remote file for input.                          !!!    &
        CALL OPRNFW BY REF(INPLUN%,ISTAT%(),NODNAM$,IDENT$,IFIL$,       &
               ICHARS$,LNTH%,BLOCK%)    ! Open remote file for input    &
\       LOC1 = 2                        ! Origin of CALL for subroutine &
\       GOSUB 250                       ! Check status

130     !!! Open remote file for output.                         !!!    &
        CALL OPWNFW BY REF(OUTLUN%,ISTAT%(),NODNAM$,IDENT$,OFIL$,       &
```

(continued on next page)

```
            ICHARS$,LNTH%,BLOCK%)   ! Open remote file for output   &
\       LOC1 = 3                    ! Origin of CALL for subroutine &
\       GOSUB 250                   ! Check status

140     !!!     MAIN LOOP - read from input file              !!!     &
        !!!                   and write to output file        !!!     &
        FOR I%=1% TO 100%

150     CALL GETNFW BY REF(INPLUN%,ISTAT%(),256%,IARRAY%())           &
                                    ! Read a record from input file

160     IF ISTAT%(0%)<>1% AND ISTAT%(1%)<>EOF% THEN 210               &
                                    ! If error, print message       &
        ELSE LNTH%=ISTAT%(1%)       ! Save no. of bytes transferred

170     !!! Check for eof                                     !!!     &
        IF ISTAT%(0%)<>1% AND ISTAT%(1%)=EOF% THEN EOFFG%=TRUE%       &
          ELSE EOFFG%=FALSE%        ! Set flag if end of file

180             IF EOFFG%=TRUE% THEN 230!IF END OF FILE, CLOSE FILES  &
                ELSE    CALL PUTNFW BY REF(OUTLUN%,ISTAT%(),LNTH%     &
                        ,IARRAY%())     ! Write record

190             IF ISTAT%(0%)<>1% AND ISTAT(1%)<>EOF% GO TO 220       &
                ELSE A%=1%              ! If unsuccessful, print message
200     NEXT I%                         ! Terminate loop

210     !!! Read error occurred                               !!!     &
        PRINT "Read error. Status = ";ISTAT%(0%);",";ISTAT%(1%)       &
\       GO TO 230                       ! Close both files

220     !!! Write error occurred                              !!!     &
        PRINT "Write error. Status = ";ISTAT%(0%);",";ISTAT%(1%)

230     !!! Close files                                       !!!     &
        FOR J%=1% TO 2%                 ! Close files 1 and 2         &
\         CALL CLSNFW(J%,ISTAT%())      ! Close each file             &
\       NEXT J%                         ! Terminate loop              &
\       IF EOFFG%<>TRUE% THEN 240       ! If flag not true, transfer not&
                                        !   successful                &
        ELSE PRINT "End of file reached. File closed"                 &
                                        ! Indicate transfer successful

240     GOTO 270                        ! Branch to end

250     !!! Subroutine to check status on completion of OPEN calls !!!  &
        IF ISTAT%(0%)=1% THEN 260       ! If success, just return       &
        ELSE PRINT "Open error. Status = ";ISTAT%(0%);",";ISTAT%(1%)    &
\         PRINT "Loc = ";LOC1           ! Origin of call                &
\           GO TO 270                   ! Quit if unsuccessful

260     RETURN                          ! Exit from subroutine

270     END                             ! End execution
```

3.10 FORTRAN Task Control

This section contains descriptions and usage guidelines for the FORTRAN task control calls summarized in Table 3–4. Task control allows you to run or abort specific tasks according to time schedules that you define in a DECnet call.

Before you issue any of these calls you must access the network by issuing an OPNNTW call. When you complete task control operations, you must issue the CLSNTW call to stop accessing the network.

3.10.1 Waiting for Requests

All calls are synchronous and pass control back to the user task only after the operation completes.

Table 3–4: FORTRAN Task Control Calls

Call	Function
ABONCW	Abort an executing task or cancel a schedule task
BACUSR	Build account and user ID information area
RUNNCW	Execute an installed task in a remote node

3.10.2 RSX Remote Task Control Utility

In order for these calls to execute successfully, the RSX Remote Task Control utility (TCL) must be installed on the remote node. If TCL is not installed, the call completes with an error.

ABONCW
(Abort an Executing Task or Cancel a Scheduled Task)

3.10.3 ABONCW — Abort an Executing Task or Cancel a Scheduled Task

Use:

Call ABONCW to abort an executing task or cancel a scheduled task.

Format:

CALL ABONCW (*lun*,[*status*],*ndsz*,*ndnm*,*passwdsiz*,*passwd*, *tsksiz*,*tsknam*,[*ident*],[*mask*])

Arguments:

lun

specifies an integer variable or constant and must be a logical unit number not currently in use.

* *status*

specifies an integer array containing the following completion status information on return from ABONCW:

status(1) Returns an error/completion code

status(2) If the error code in *status*(1) indicates a network reject (–7), *status* (2) contains the disconnect or reject reason code. Refer to Appendix A. Otherwise, *status*(2) contains a directive error code (if *status*(1) is –40) or null value (0).

ndsz

specifies an integer variable or constant containing the node name length in bytes.

ndnm

specifies a 1- to 6-element byte array containing the name of the target node.

passwdsiz

specifies an integer variable or constant containing the password length in bytes.

passwd

specifies an array containing a user password with which to gain access to the remote system. Specify an array size consistent with the connect block size that you specified in the OPNNT call. The password can have 1–8. bytes in a short connect block, or 1–39. bytes in a long connect block.

A privileged password lets a user abort any task running on the remote node without specifying the *ident* parameter. A nonprivileged password lets a user abort a task only by specifying the correct *ident* parameter.

tsksiz

specifies the remote task name length in bytes.

tsknam

specifies a 1- to 6-element byte array containing the name of the remote task to abort or cancel.

* *ident*

specifies an integer variable containing the negated task control block address of the remote task. This value is returned to the *ident* parameter of the RUNNCW call when the RUNNCW call completes. This argument is optional for a user with a privileged password.

* *mask*

indicates how ABONCW is used. This argument is optional.

Omitting the *mask* argument or specifying a value of 0 aborts only the executing task that the call specifies. Specifying the value 1 cancels the rescheduling of the specified task and continues execution of the current active task. Specifying a value greater than 1 aborts the executing task and cancels the rescheduling of the task.

Error/Completion Codes:

1	The call completed successfully.
-1	System resources needed for the logical link are not available.
-7	The connection was rejected by the network. Refer to Appendix A.
-8	A logical link has already been established using this LUN.
-9	The task is not a network task: OPNNT did not execute successfully.
-21	The requested task is not installed on the remote node.
-23	An ABONCW was issued for a task that was not active.
-24	A privileged violation has occurred. You are not a privileged user, and you are attempting an ABONCW for a task with improper identification.
-25	An ABONCW was issued for a task that either was being loaded into or was exiting from the remote node.
-40	A directive error has occurred. Directive error codes are defined in the *RSX-11M/M-PLUS Executive Reference Manual.*

BACUSL
(Build Account and User ID Information Area (Long))

3.10.4 BACUSL — Build Account and User ID Information Area (Long)

Use:

Call BACUSL in the source task to build the user ID and account areas of the outgoing connect block for task control programming.BACUSL supports 39.-character user IDs and accounts.

BACUSL's function is similar to BACCL's, but you do not specify a password with BACUSL. Instead, you include the password with the ABONCW or RUNNCW call.

If you have defined an alias node name that includes explicit access control information, or use proxy access, you need not call BACUSL.

Format:

CALL BACUSL ([*status*],[*usersz*,*user*],[*accnosz*,*accno*])

Arguments:

* *status*

specifies an integer variable. On return from BACUSL, this optional argument is set to .TRUE.(−1) if the call completed successfully. It is set to .FALSE.(0) if one of the arguments to BACUSL is invalid.

usersz

specifies an integer variable or constant containing the user ID length in bytes.

user

specifies a 1- to 39.-byte array containing the user ID. The arguments *usersz* and *user* are paired optional arguments. Include both or omit both.

accnosz

specifies an integer variable or constant containing the length in bytes of the account number. RSX target systems do not use this argument.

accno

specifies a 1- to 39.-byte array containing the account number. The *accnosz* and *accno* arguments are paired optional arguments. Include both or omit both.

BACUSR
(Build Account and User ID Information Area (Short))

3.10.5 BACUSR — Build Account and User ID Information Area (Short)

Use:

Call BACUSR in the source task to build the user ID and account areas of the outgoing connect block for task control programming. BACUSR supports16.-character user IDs and accounts.

BACUSR's function is similar to BACC's, but you do not specify a password with BACUSR. Instead, you include the password with the ABONCW or RUNNCW call.

If you have defined an alias node name that includes access control information, or use proxy access, you need not call BACUSR.

Format:

CALL BACUSR ([*status*],[*usersz*,*user*],[*accnosz*,*accno*])

Arguments:

* *status*

specifies an integer variable. On return from BACUSR, this optional argument is set to .TRUE.(−1) if the call completed successfully. It is set to .FALSE.(0) if one of the arguments to BACUSR is invalid.

usersz

specifies an integer variable or constant containing the user ID length in bytes.

user

specifies a 1- to 16.-byte array containing the user ID. The arguments *usersz* and *user* are paired optional arguments. Include both or omit both.

accnosz

specifies an integer variable or constant containing the length in bytes of the account number. RSX target systems do not use this argument.

accno

specifies a 1- to 16.-byte array containing the account number. The *accnosz* and *accno* arguments are paired optional arguments. Include both or omit both.

RUNNCW
(Execute an Installed Task in a Remote Node)

3.10.6 RUNNCW — Execute an Installed Task in a Remote Node

Use:

RUNNCW allows you to execute an installed task in a remote node using any or all of the following options:

- Execute the task immediately.
- Schedule the task for execution at some future time.
- Schedule the task for periodical execution based on predefined time schedules.

Format:

CALL RUNNCW (*lun*,[*status*],*ndnm*,*passwdsz*,*passwd*,*tsksz*,*tsknam*,[*ident*], [*uic*],[*smg*,*snt*],[*rmg*,*rnt*])

Arguments:

lun

specifies an integer variable or constant and must be a logical unit number not currently in use.

* *status*

specifies an integer array containing the following completion status information on return from RUNNCW:

status(1) Returns an error/completion code

status(2) If the error code in *status*(1) indicates a network reject (−7), *status* (2) contains the disconnect or reject reason code. Refer to Appendix A. Otherwise, *status*(2) contains a directive error or is not used.

ndsz

specifies an integer variable or constant containing the node name length in bytes.

ndnm

specifies a 1- to 6-element byte array containing the name of the target node.

passwdsiz

specifies an integer variable or constant containing the password length in bytes.

passwd

specifies an array containing a user password with which to gain access to the remote node. Specify an array size consistent with the connect block size that you specified in the OPNNT call. The password can have 1–8. bytes in a short connect block, or 1–39. bytes in a long connect block.

A privileged password lets you run a task under any user identification code on the remote node. A nonprivileged password lets you run a task under only the UIC assigned to you.

tsksiz

specifies an integer variable or constant containing the remote task name length in bytes.

tskname

specifies a 1- to 6-element byte array containing the name of the remote task to execute.

* *ident*

*

specifies an integer variable containing the negated task control block address of the remote task when RUNNCW completes. ABONCW uses this value. If you do not plan to cancel or abort this task later, you can omit this argument.

uic

specifies a 2-byte array containing the group and user codes under which the task runs on the remote node. The first element of the array contains the user member code; the second element contains the user group code. This argument is optional with a privileged password. If a privileged user omits this argument, the task runs under its default UIC on the remote node.

smg

specifies an integer variable or constant containing the schedule delta magnitude. The value of this optional argument is the difference in time from the issuance of the call to the time the task is to run at the remote node.

This argument is used with the following argument, *snt*, which specifies the unit of time used to schedule the task (in hours, minutes, seconds, or ticks). In no case can the magnitude exceed 24 hours.

snt

specifies an integer variable or constant containing the schedule delta unit. This argument is a code identifying the time unit specified with the *smg* argument. The time unit codes are as follows:

Code	Description
1	Ticks: A tick occurs for each clock interrupt and depends on the type of clock installed in the system. Line frequency clock: The tick rate is either 50 or 60 per second and corresponds to the powerline frequency. Programmable clock: A maximum of 1000 ticks per second is available. The exact rate is determined at system generation.
2	Seconds
3	Minutes
4	Hours

rmg

specifies an integer variable or constant containing the reschedule delta magnitude. The reschedule interval is the difference in time from task initiation to the time the task is to be reinitiated on the remote node. The task is executed each time the elapsed time equals the reschedule magnitude specified in this argument. If this time interval elapses and the task is still active, no reinitiation request is issued. However, a new reschedule interval is started.

This argument is used with the following argument, *rnt*, which specifies the unit of time used to reschedule the task (in hours, minutes, seconds, or ticks). In no case can the magnitude exceed 24 hours.

rnt

specifies an integer variable or constant containing the reschedule delta unit. This argument is a code identifying the time unit to use with the delta magnitude specified in the *rmg* argument.

NOTE

- If you omit the *smg*, *snt*, *rmg*, and *rnt* arguments, the task is executed immediately.
- If you specify *smg* and *snt*, but omit *rmg* and *rnt*, the task is executed once at the scheduled time.
- If you specify *rmg* and *rnt*, but omit *smg* and *snt*, the task is executed immediately and again each time the reschedule delta time has elapsed.
- You can specify all four arguments. For example:

  CALL RUNNCW (*lun*,*status*,*ndsz*,*tsksiz*,*tsknam*, *uic*,1,4,4,4)

  specifies that the task runs for the first time in one hour and then every four hours after that.

Error/Completion Codes:

1	The call completed successfully.
-1	System resources needed for the logical link are not available.
-7	The connection was rejected by the network. Refer to Appendix A.
-8	A logical link has already been established using this LUN.
-9	The task is not a network task: OPNNT did not execute successfully.
-20	There is insufficient dynamic memory on the remote node.
-21	The requested task is not installed on the remote node.

-22 RUNNCW has an invalid time parameter.

-23 An RUNNCW call was issued without scheduling parameters for a task that is already active.

-24 A privileged violation has occurred. You are not a privileged user, and you are attempting to issue a RUNNCW under a UIC different from the UIC to which you are assigned on the remote node.

-26 A RUNNCW was issued under an invalid UIC (for example, [1,0] or [0,1]).

-40 A directive error has occurred. Directive error codes are defined in the *RSX-11M/M-PLUS Executive Reference Manual*.

3.10.7 FORTRAN Task Control Programming Example

The RUNABO.FTN program uses DECnet task control calls to run or abort a task on a specified local or remote node. After executing your task control request, the program prompts you to enter another request to run or abort the associated task. When you finish entering task control requests, press CTRL/Z to exit from the request-prompting loop and stop the program.

Before running RUNABO.FTN, you must install the TCL task on the target node.

This programming example is included in your tape or disk kit.

```
C
C   RUNABO.FTN - Run or abort a task installed on remote node
C
C
C Copyright (C) 1983, 1985, 1986, 1987 by
C Digital Equipment Corporation, Maynard, Mass.
C
C
C This software is furnished under a license and may be used and  copied
C only  in  accordance  with  the  terms  of  such  license and with the
C inclusion of the above copyright notice.  This software or  any  other
C copies  thereof may not be provided or otherwise made available to any
C other person.  No title to and ownership of  the  software  is  hereby
C transferred.
C
C The information in this software is subject to change  without  notice
C and  should  not  be  construed  as  a commitment by Digital Equipment
C Corporation.
C
C Digital assumes no responsibility for the use or  reliability  of  its
C software on equipment which is not supplied by Digital.
C
C
C   This program illustrates the DECnet RSX task control routines.
C
C   To task build, use the following command string:
C
C   >TKB RUNABO,RUNABO=RUNABO,LB:[1,1]NETFOR/LB,F4POTS/LB
C
C   Note: The TCL task must be installed on the specified node.
C
C
      LOGICAL*1 ANSWER,RUN,ABO,TARTSK(6),TARNOD(6),PASSWD(8),USERID(16)
      LOGICAL*1 ACCNT(16)
      INTEGER STATUS(2),STAT
      INTEGER*2 MSTAT(3),IDENT
      DATA RUN/'R'/,ABO/'A'/
```

(continued on next page)

```
C
C  Create the network data queue
C
        CALL OPNNTW(1,STATUS,MSTAT)
        IF (STATUS(1) .NE. 1) WRITE(5,8)STATUS(1)
C
C  Prompt for target node and target task
C
  10    WRITE(5,1)
   1    FORMAT(5X,$'Enter target node: ')
        READ(5,2,END=999)ICNT1,TARNOD
   2    FORMAT(Q,16A1)
C
        WRITE(5,3)
   3    FORMAT(5X,$'Enter target task: ')
        READ(5,2,END=999)ICNT2,TARTSK
C
C Prompt for access control information
C
        WRITE(5,50)
  50    FORMAT(5X,$'Enter target user ID: ')
        READ (5,2,END=999)ICNT4,USERID
C
        WRITE(5,4)
   4    FORMAT(5X,$'Enter target password: ')
        READ (5,2,END=999)ICNT3,PASSWD
C
        WRITE(5,11)
  11    FORMAT(5X,$'Enter target account number: ')
        READ (5,2,END=999)ICNT5,ACCNT
C
        WRITE(5,6)
   6    FORMAT(5X,$'Enter RUN (R) or ABORT (A): ')
        READ(5,7,END=999)ANSWER
   7    FORMAT(A1)
C
C  Decide whether to call BACUSR
C
        IF (ICNT4 .EQ. 0 .AND. ICNT5 .EQ. 0) GO TO 70
        CALL BACUSR (STAT,ICNT4,USERID,ICNT5,ACCNT)
        IF (STAT .EQ. .TRUE.) GO TO 70
        WRITE(5,80)STAT
  80    FORMAT (' Error building connect block ')
        GOTO 10
C
C  Decide whether to run or abort the task
C
  70    IF (ANSWER .EQ. ABO) GOTO 20
        IF (ANSWER .EQ. RUN) GOTO 30
        GOTO 999
C
C  Abort the task and print status
C
  20    WRITE (5,100)
 100    FORMAT (5X,$'IDENT of task to abort (0 if password is privileged): ')
        READ (5,110)IDENT
 110    FORMAT(I6)
        CALL ABONCW (2,STATUS,ICNT1,TARNOD,ICNT3,PASSWD,ICNT2,TARTSK,IDENT)
```

(continued on next page)

```
C
      WRITE(5,8)STATUS(1)
    8 FORMAT(' Status = ',I7)
      GOTO 10
C
C  Run the task and print status.
C
   30 CALL RUNNCW(2,STATUS,ICNT1,TARNOD,ICNT3,PASSWD,ICNT2,TARTSK,IDENT)
      WRITE (5,8)STATUS(1)
      IF (STATUS(1) .EQ. 1) WRITE (5,90) IDENT
   90 FORMAT (' The IDENT is ',I6)
      GOTO 10
  999 STOP
      END
```

4

DLX Ethernet Programming Facilities

The Direct Line Access Controller (DLX) gives application programs a direct interface to the data link, bypassing the standard DECnet user interface. With DLX, you can communicate with DECnet or non-DECnet based systems. Because DLX does not offer higher-level DECnet services, such as routing and guaranteed delivery, it can give high performance in network applications. DLX also lets you build customized user-level protocols that best suit your applications.

To use DLX, you issue queued input/output (QIO) calls to the NX: device. Your DLX program uses the Ethernet and/or IEEE 802.3 standard. It can communicate with a DLX program or the equivalent data link function on an adjacent DECnet–RSX or non-DECnet node. Your DECnet–RSX node can simultaneously run multiple DECnet and DLX tasks, each possibly communicating with different remote nodes.

DLX is automatically built for DECnet–RSX–11M–PLUS and DECnet–Micro/RSX systems; it is optional for DECnet–RSX–11M. It is also optional for RSX–11S systems, but is required on a host for down-line loads and up-line dumps from RSX–11S systems. You can use DLX to communicate over all devices that DECnet–RSX supports. For information on programming for point-to-point and multipoint lines, refer to Chapter 5.

Throughout this chapter, the term "the Ethernet" refers to the physical transmission media (cables and controllers) and data link level software that provides access to the physical media according to a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. The physical channel may be broadband or baseband. The Ethernet can transmit frames in formats that conform to either the Ethernet standard or the IEEE standards.

4.1 Preparing the System

Before your system runs a DLX program, the DLX process must be loaded and the line set.

The person in charge of network or system management installs the network, usually by executing a command file that contains the command for loading DLX. When DLX is loaded, it resides in the common partition NT.DLX.

The network manager also sets the line, either by answering Yes to the NETGEN question that asks about marking the line for load, or by issuing the Network Control Program (NCP) SET LINE command. For information on using NCP to set the line, refer to the *DECnet–RSX Guide to Network Management Utilities*.

4.2 Including Higher-Level Services

DLX programming requires a thorough knowledge of MACRO–11 assembly language and experience in writing real-time application programs.

Since DLX bypasses the higher levels of DECnet you lose the services at those levels and must, therefore, include them in your application. Your programs must provide the following:

Flow control	DLX does not support flow control for data transfer. The DLX programs that run on different nodes must therefore synchronize with each other before transferring data. If the tasks are unsynchronized, data can be lost.
Error recovery	The DLX software reports errors, but your program must include error recovery procedures.
Data segmentation	When transmitting data, your program must segment it; the buffer size must be appropriate to the controller devices on the communicating systems. For information on appropriate buffer sizes, consult your network manager.

Note that all incoming and outgoing DLX and DECnet messages are buffered in a shared network buffer pool. Your network manager can increase the size and/or number of buffers to maintain good throughput performance, if necessary. For information on displaying and setting buffer sizes, refer to the DECnet–RSX network management documentation.

Also note that you must use the /PR:0 switch to task build your DLX programs.

4.2.1 Using DLX Resources

DLX provides macros and QIOs to use in your application.

The DECnet macro library, NETLIB.MLB, defines the offsets and macros that DLX QIOs use. During NETGEN, this library is transferred to your system. The definition macro DLXDF$ contains definitions for offsets and macros.

Your program must issue an .MCALL statement and explicitly invoke the definition macro, as in the following example:

```
.MCALL DLXDF$  ; extract from macro library
         .
         .
         .
      DLXDF$  ; define DLX symbols
```

You can use the following QIO functions in Ethernet programming:

IO.XOP	Open a port on the Ethernet device. This lets your program treat the Ethernet as a device that your QIOs control.
IO.XSC	Set characteristics for your Ethernet port. You can set such port characteristics as the frame format to use, the addresses from which you want to receive messages, and so forth.
IO.XGC	Get characteristics of the port.
IO.XTM	Transmit a message.
IO.XRC	Ready the port to receive a message.
IO.XCL	Close the port and relinquish use of the controller.

4.3 Using DLX to Access the Ethernet

An Ethernet data link on a single Ethernet controller supports multiple concurrent users. Each station represents an available port onto the Ethernet channel.

Because multiple users simultaneously access the Ethernet channel, your program must use addressing mechanisms that ensure delivery of messages to the correct recipient. Any message that you transmit on the Ethernet must include an Ethernet address that identifies the target node. The message must also include an additional identifier that directs the message to the correct user on the target node; this identifier varies according to the frame format you choose to use. DLX lets you choose to build frames according to the Ethernet or IEEE 802.3 standard, or both.

The Ethernet format is a proprietary standard that belongs to Digital Equipment, Intel, and Xerox corporations. The IEEE 802.3 format, in contrast, is a standard for multi-vendor networking. To communicate with other Digital nodes, you might have applications that use the Ethernet frame format, and to communicate with non-Digital nodes, you might use the 802.3 frame format. Any single application can send and receive both types of frames. Later sections of this chapter describe how to set up the Ethernet address and use each frame format.

This chapter covers the following topics:

- Synchronizing DLX programs
- Using physical and multicast addressing
- Setting up the Ethernet address
- Using characteristics buffers
- Processing Ethernet-format frames
- Processing 802.3-format frames
- DLX QIOs

References throughout the chapter to the IEEE standard are to the information in two publications listed in the Preface to this manual: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications* (802.3), and *Logical Link Control* (802.2).

4.3.1 Synchronizing DLX Programs

In writing your DLX application, you must synchronize the programs on both nodes to ensure that they can cooperate. Both communicating nodes must do the following:

- Open the line, specifying the same frame format (Ethernet or 802.3).
- Specify what frames you want the port to send and receive. For Ethernet frame format, both nodes must enable the port to send and receive frames with the same protocol type. For 802.3 frame format, both nodes must enable the port to send and receive frames with the same Subnetwork Access Protocol (SNAP) identifier, or to receive frames with each other's Service Access Point (SAP). Later sections explain how you use protocol types, SNAPs, and SAPs.

- The nodes must then coordinate their transmission and reception. The receiving node must have a receive QIO pending before the sending node transmits.

4.3.2 Using Physical and Multicast Addressing

You can transmit and receive messages over the Ethernet in physical or multicast address mode. Physical addressing sends messages to a single destination node. Multicast addressing sends messages to a group of nodes. If each node in the group enables reception of messages with a given multicast address, a single transmission to that address can reach all nodes in the group.

To send messages in physical mode, you specify the destination address in a transmit request. If the target node is a non-DECnet node, you send messages to its Ethernet hardware address. If the target node is a DECnet node, you send messages to its Ethernet physical address. The Ethernet physical address is derived from the node address. The next section explains how to set up the Ethernet address for remote DECnet nodes. You receive any physical mode messages that other nodes address to your Ethernet hardware address on a non-DECnet node, or to your Ethernet physical address on a DECnet node. You need not specially enable these addresses.

To send multicast messages, you simply specify the Ethernet multicast address as the destination on a transmit operation. Any node can send messages to any multicast address. To receive messages sent to a multicast address, you specify the address when setting port characteristics with IO.XSC. You can receive any number of multicast addresses.

The multicast address for Digital Equipment Corporation customer use is 09–00–2B–00–00–0F. In a Digital-only environment, you can use other numbers that fall outside the range of those reserved for internal Digital use. In a multi-vendor environment, other multicast addresses might conflict with the other vendors' conventions. For more information on multicast addresses, refer to Appendix G.

For a further description of Ethernet addressing, refer to *DECnet–RSX Network Management Concepts and Procedures*.

4.3.3 Setting Up the Ethernet Address

All messages on an Ethernet channel have one 48-bit Ethernet address that specifies the destination node and one that specifies the source node. While you need not supply your own (source) address when you transmit, you must always supply the destination address.

When sending messages to a DECnet node, you can derive the Ethernet address from the node address. The DECnet destination consists of 6 bytes. The first four bytes are standard, and contain the following octal values:

Byte 0	252
Byte 1	0
Byte 2	4
Byte 3	0

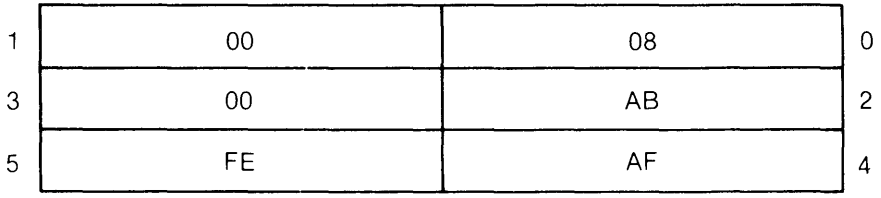
For bytes 4 and 5, use an octal version of the area number and node number, and format them as follows:

Bits 10 to 15	DECnet area number (The default area number is 1.)
Bits 0 to 9	DECnet node number

For example, you convert DECnet node addresses of 1.154 and 4.153 to destination addresses as follows:

Node Address (Decimal)	Destination Address (Octal)	Hexadecimal Equivalent
1.154	252,0,4,0,4,232	AA-00-04-00-04-9A
4.153	252,0,4,0,20,231	AA-00-04-00-10-89

To send messages to a non-Digital node, you must know the destination hardware address. Be sure to program the address into the correct bytes. For example, if the destination address, in hexadecimal notation, is 08-00-AB-00-AF-FE, enter the hexadecimal values as follows:



LKG-1039-87

4.3.4 Setting Up a Characteristics Buffer

Most of the DLX QIOs let you specify or read characteristics for the QIO. Some characteristics affect the Ethernet port; others affect a specific transmit or receive operation. You can set characteristics for the port when you issue IO.XOP to open the port and by issuing the Set Characteristics QIO. You can read the port characteristics by issuing the IO.XGC (get characteristics) QIO. You can also set certain characteristics for each transmit (IO.XTM) and receive (IO.XRC) QIO.

Some characteristics are required for a QIO; others are optional. A transmit QIO, for example, always requires that you supply the destination Ethernet address characteristic.

To set or read characteristics, you create a characteristics buffer and enter the buffer's address and length as QIO parameters. One or more characteristics blocks in the buffer specify the characteristics to set or read. This chapter describes the various characteristics blocks for Ethernet programming in the section about processing Ethernet frames, and the characteristics for 802.3 programming in the section about processing 802.3 frames. In addition, each QIO description includes a description of the characteristics to use with that QIO.

For example, you can specify that a port opens in Ethernet or 802.3 frame format when you issue the Open Port (IO.XOP) directive. You create a buffer that includes the frame format characteristics block, CC.FMO, into which you enter the frame format value. You then reference the buffer's address and length in the QIO. Once you have specified the frame format (Ethernet format is the default), you can then specify other format-specific characteristics. For 802.3 format, you can specify the 802.3 service class; for Ethernet format, you can specify the Ethernet protocol type. You append the blocks for these characteristics to the frame format block in the buffer. An example later in this section illustrates how a program opens a port for 802.3 format and specifies the service class.

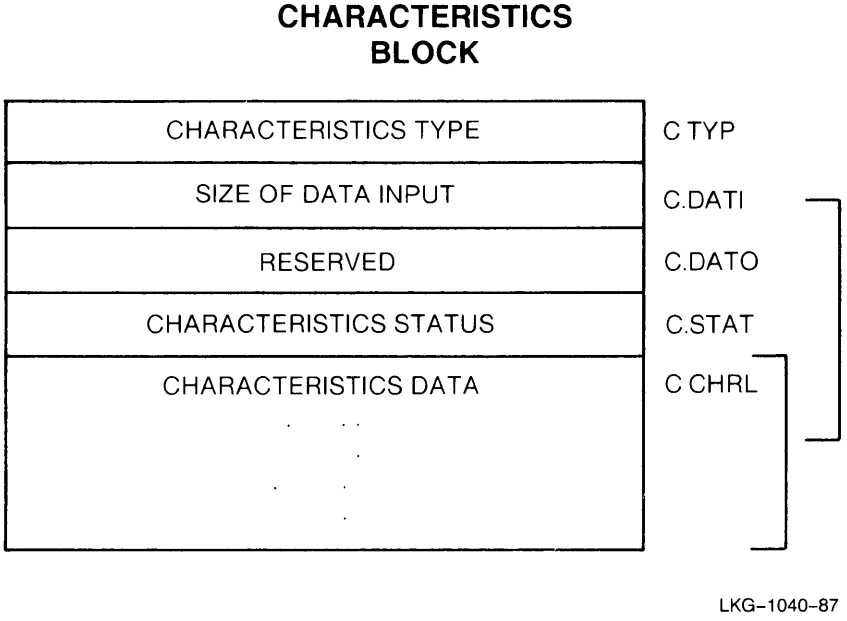
A characteristic that affects the port affects all data that the port handles, unless you override the characteristic on a specific transmit or receive QIO. In contrast, a characteristic that affects a transmit or receive request affects only the specific transmit or receive QIO.

Characteristics blocks have a standard format in the first four fields. Table 4–1 describes these fields.

Table 4–1: The First Four Fields in a Characteristics Block

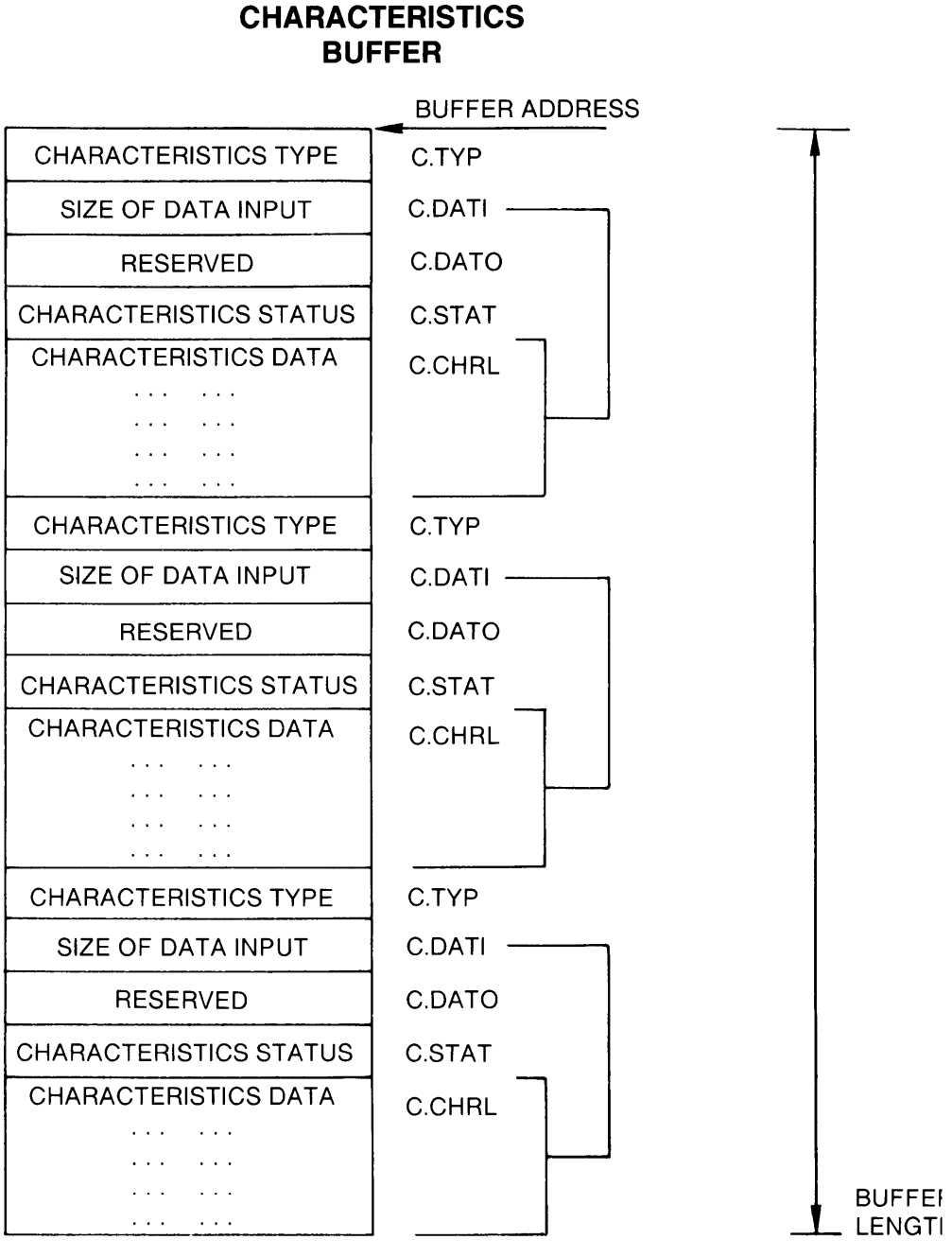
Word	Name	Contents	Use
0	C.TYP	Characteristics type	Identifies what characteristic the block contains information about. For example, CC.ADR in this field specifies a destination address block, CC.MCT specifies a multicast address block, and so on.
1	C.DATI	Size of data input	Identifies the length, in bytes, of the characteristics data in the C.CHRL field. For example, if a CC.MCT block contains one 6-byte multicast address, this field contains a 6.
2	C.DATO	Reserved for size of data output	Gives the length, in bytes, of any returned data on completion of characteristics processing. Always put a zero (0) in this field.
3	C.STAT	Reserved for characteristics status	Contains a code indicating completion status after characteristics block processing. Always put a zero (0) in this field.

The characteristics block for a single characteristic looks like this:



The fifth field, C.CHRL, contains the specific characteristics information.

You can append multiple characteristics blocks in a single characteristics buffer as follows:



LKG-1041-8

The following program fragment uses a characteristics buffer in opening an Ethernet port. Before issuing the IO.XOP QIO, the program creates a characteristics buffer with two characteristics blocks. As an example, the first block requests 802.3 frame format. With 802.3 frame format, the data link provides certain services if you request Class I service. The second block requests Class I service. The IO.XOP QIO then references the buffer that contains these characteristics blocks.

```
..
          .MCALL    DLXDF$
..
          DLXDF$                  ; Define DLX I/O codes and symbols

PRTLUN    = 1                     ; Logical unit number of port
PRTFLG    = 1                     ; Event flag for all port I/O
DEVNM:    .ASCII    \UNA-0\       ; Address of device name
DEVLN     = .-DEVNM               ; Length of device name
          .EVEN
IOSB:     .BLKW     2            ; I/O status block
..
;
; Characteristics buffer for open
;
; This buffer contains a pair of characteristics blocks - one to
; open the port for handling 802.3 frames and the other to
; request Class I service for the 802.3 port.

OPNCHB:
;  Define frame format for open
          .WORD     CC.FMO      ; C.TYP  Characteristic type
          .WORD     2           ; C.DATI Frame format takes 2 bytes
          .WORD     0           ; C.DATO To be returned
          .WORD     0           ; C.STAT To be returned
          .WORD     NX$802      ; C.CHRL 802 frame format
;  Define Class I service for open
          .WORD     CC.SCO      ; C.TYP  Characteristic type
          .WORD     2           ; C.DATI Service class takes 2 bytes
          .WORD     0           ; C.DATO To be returned
          .WORD     0           ; C.STAT To be returned
          .WORD     NX$CLI      ; C.CHRL 802 Class I service
OPNCHL    =.-OPNCHL             ; Size of characteristics buffer
..
;
; Directive parameter blocks
;
OPNDPB    QIOW$
     IO.XOP,PRTLUN,PRTFLG,,IOSB,,<DEVNM,DEVNML,OPNCHB,OPNCHL>
..
```

In the line containing the IO.XOP call, OPNCHB specifies the address of the characteristics buffer and OPNCHL specifies its length.

The order in which your program references the characteristics can be important. DLX processes the characteristics buffer sequentially, and some characteristics create prerequisite conditions that are prerequisites for other characteristics. For instance, if the previous example attempted to request Class I service without first requesting 802.3 frame format, an error would occur.

The *DECnet–RSX Network Management Concepts and Procedures* manual has more information on using Ethernet devices. For detailed information on 802.3 frame formats, refer to the IEEE standard. The next sections describe special procedures for processing Ethernet and 802.3 frame format.

4.3.5 Processing Ethernet Frames

This section describes special considerations for using Ethernet frame format on the Ethernet channel.

All Ethernet frames contain a 16-bit identification number called an Ethernet protocol type. When a message arrives at the controller, the protocol type identifies which port receives the frame. DLX applications that communicate across the Ethernet must always enable the same Ethernet protocol type.

You enable the port to receive the protocol type by issuing an IO.XSC to set the port characteristics. You specify the protocol type in a CC.DST characteristic block, which IO.XSC references. Enable the protocol type after opening the port, but before receiving or transmitting messages with the protocol type. Specify an enabled protocol type in every transmit QIO and read it in every receive QIO.

The protocol type for Digital Equipment Corporation customer use is 60–06. For more information on protocol type values, refer to Appendix G.

When enabling an Ethernet protocol type, you can add the following information to the CC.DST characteristics block to further specify how the port handles the protocol:

- Protocol flags that specify modes for receiving and sending messages with the protocol type.
- Specific remote addresses to and from which the port will process messages with the protocol type.

4.3.5.1 Setting Protocol Flags

The characteristic block that you use to enable a protocol type has a field for protocol flags. Protocol flags can do the following:

- Set the receipt mode for the protocol type to exclusive or default
- Request padding support for frames with the protocol type
- Disable the protocol type

Your Ethernet port can receive messages with a given protocol type in exclusive, default, or normal usage mode:

Exclusive LF$EXC	The application receives all messages with the protocol type.
Default LF$DEF	The application receives messages with the protocol type from any address that another application does not receive in
Normal	The application specifies the addresses from which to receive messages with the protocol type.

For exclusive or default mode, you set the appropriate flag in the CC.DST characteristics block. Normal mode requires that you omit the flag and specify addresses in the characteristic block, as the following section describes.

To request padding support, you can set the LF$PAD flag. Padding is a highly recommended option for ensuring data integrity. On transmit operations, the device driver software pads each frame to the minimum Ethernet size; the actual data in a frame may therefore be less than the frame's physical length. With padding support, however, the data link will prefix the actual data with a 2-byte length field on transmit and read the length field on receive. On receive, it strips the padding from the message before passing the message to your application. You must set the flag to get padding support when you enable a protocol.

To disable a protocol type, set the LF$DIS flag.

4.3.5.2 Specifying Protocol/Address Pairs

To instruct the port to process the protocol type to and from certain addresses only, add the addresses to the characteristics block that enables the protocol type (CC.DST). To ensure communication among DLX programs on different nodes, each program enables the same protocol type. Each program pairs the protocol with the remote nodes' physical or multicast Ethernet addresses or sets the protocol flag to receive in exclusive mode.

Two or more users on a node can enable the same protocol or the same addresses, but only one user can enable any protocol/address pair. This prevents two programs on a node from competing for the same frames.

4.3.5.3 Using Characteristics Blocks

Table 4–2 lists the characteristics for Ethernet frame format according to the DLX function and QIO with which you use them. Within each function, the characteristics are listed alphabetically. Each QIO description later in this chapter has detailed information on the related characteristics.

Table 4–2: Characteristics for Ethernet Frame Format

Symbol & Value	What It Does
Open the Port (IO.XOP)	
CC.FMO (103)	Defines a single frame format for the port.
Set and Get Port Characteristics (IO.XSC and IO.XGC)	
CC.DST (200)	Enables the port to send and receive messages with a specified protocol type and pairs the protocol with addresses on a set characteristics operation; returns the protocol type and addresses on a get characteristics operation.
CC.FRM (202)	Specifies a second frame format for the port on a set characteristics operation; returns the second frame format on a get characteristics operation.
Transmit and Receive (IO.XTM and IO.XRC)	
CC.ADR (100)	Sets the destination node address on transmit; returns the source node address on receive.
CC.DAD (102)	Returns the destination address to which a received message was sent.
CC.FMM (105)	Sets the frame format of a message on transmit; returns the frame format on receive.
CC.PRO (101)	Sets the protocol type of a message on transmit; returns the protocol type on receive.

4.3.6 Processing IEEE 802.3 Frames

The Institute of Electrical and Electronics Engineers (IEEE) has defined the 802.3 frame format for communicating over the Ethernet. You may choose to use 802.3 frame format, especially for inter-vendor communications. Using 802.3 format, a DLX program on a Digital node can communicate with a similar program on a Digital or non-Digital node. To use 802.3 frame format, familiarize yourself with the IEEE standard.

When you use 802.3 format, you must choose the 802.3 service class and addressing mode to use. The next sections explain your choices.

4.3.6.1 Specifying the Service Class

The service class determines the level of service that the data link provides to your application. DECnet–RSX supports two 802.3 service classes: Class I and user-supplied service.

Class I service lets your program perform IEEE 802.3 Type I operations and it supports three frame types:

UI Unnumbered information

TEST Test

XID System identification

UI frames contain data to send and receive. TEST and XID frames verify that a node with which you want to communicate is up and running the correct software. The IEEE standard fully describes these frame types.

With Class I service, the data link:

- Filters out all extraneous types of messages.
- Handles unsolicited XID and TEST messages from other nodes.
- Builds and strips frames, letting your program handle just the data in the frame without supplying or reading headers.

Class I service also provides a group addressing capability described in a following section.

With user-supplied service, your application can use any IEEE 802.3 frame types, but your application must build and strip them. Your program must also include routines for filtering out unwanted types of messages.

You can specify Class I service when you open the port or set port characteristics for 802.3 format. The default service type is user-supplied.

You must also choose to use either Service Access Points (SAPs) or Subnetwork Access Protocols (SNAPs) to identify your 802.3 frames. The next two sections describe SAPs and SNAPs.

4.3.6.2 Defining Service Access Points

Service Access Points (SAPs) identify each application that accesses the Ethernet in 802.3 format. The destination Ethernet address identifies the target node for the message, and then the Destination SAP (DSAP) and Source SAP (SSAP) identify the destination and source application at the port. All 802.3 frames contain SAPs, which are therefore helpful in multi-vendor programming environments.

With SAPs, you must define at least one Individual SAP (ISAP) for each 802.3 application. The application's ISAP must be unique and exclusive. You then use IO.XSC to set the port characteristics to receive messages with the specified ISAP. On transmit, you supply the Destination SAP and Source SAP; on receive, you read the DSAP and SSAP.

Each ISAP on a node identifies only one application, but each application can enable multiple SAPs. You might use different SAPs, for example, to identify different functions that the application performs.

With Class I service only, your application program can also enable one or more Group SAPs in addition to its individual SAPs. Group SAPs (GSAPs) let you send messages to a group of programs on a remote node. To define a program as a member of a group, the program sets the port characteristics to enable receipt of the GSAP. Any other application can then address messages to the entire group at once, simply by specifying the GSAP as the Destination SAP in a transmit. An application must enable at least one ISAP before receiving frames addressed to its GSAP.

Each 802.3 frame has a control (CTL) field that specifies what type of data the frame contains. With SAPs, you must specify the contents of the CTL field on every transmit and read them on every receive. With Class I service, the control field can contain the value that specifies a UI, a TEST, or an XID frame. With user-supplied service, the field can contain a value that specifies any type of 802.3 frame.

4.3.6.3 Defining SNAP Protocol Identifiers

Subnetwork Access Protocol (SNAP) identifiers provide an alternate identification mode for 802.3 frames. SNAP identifiers consist of 5 bytes. Because they are larger than SAPs, they let you address many more users on a single node, offering flexibility for uniquely identifying applications in a very large environment. SNAP identifiers might be preferable in an environment that consists only of Digital nodes.

To use SNAPs, you assign identical SNAP identifiers to communicating programs. You set characteristics to enable the port to transmit and receive messages with that SNAP identifier, using the CC.SNP characteristic with IO.XSC. Then you specify a message's SNAP identifier on transmit and read it on receive, using the CC.SNM characteristic with IO.XTM and IO.XRC. You need not specify the Destination SAP, Source SAP, or CTL fields, as you would in simple SAP addressing. DLX automatically builds these fields with standard values when you specify a SNAP identifier. When the data link receives a frame with these standard DSAP, SSAP, and CTL values, it automatically proceeds to processing the SNAP identifier.

When enabling a SNAP identifier, you can add the following information to the CC.SNP characteristics block to further specify how the port handles the protocol:

- Protocol flags that specify modes for receiving and sending messages with the protocol type.

- Specific remote addresses to and from which the port will process messages with the protocol type.

4.3.6.4 Setting Protocol Flags

The characteristics block that you use to enable a SNAP identifier has a field for protocol flags. Protocol flags can do the following:

- Set the receipt mode for the protocol type to exclusive or default

- Disable the protocol type

Your Ethernet port can receive messages with a given SNAP identifier in exclusive, default, or normal usage mode:

Exclusive LF$EXC	The application receives all messages with the SNAP identifier.
Default LF$DEF	The application receives messages with the SNAP identifier from any address that another application does not receive in normal mode.
Normal	The application specifies the addresses from which to receive messages with the SNAP identifier.

For exclusive or default mode, you set the appropriate flag in the CC.DST characteristics block. Normal mode requires that you omit the flag and specify addresses in the characteristic block, as the following section describes.

To disable a SNAP identifier, set the LF$DIS flag.

4.3.6.5 Specifying Protocol/Address Pairs

To instruct the port to process the SNAP identifier to and from certain addresses only, add the addresses to the characteristics block that enables the protocol type (CC.SNP). To ensure communication among DLX programs on different nodes, each program enables the same SNAP identifier. Each program pairs the protocol with the remote nodes' physical Ethernet addresses or sets the protocol flag to receive in exclusive mode.

Two or more users on a node can enable the same SNAP identifier or the same addresses, but only one user can enable any protocol/address pair. This prevents two programs on a node from competing for the same frames.

Appendix G has information on the available SNAP identifiers.

4.3.6.6 Using Characteristics Blocks

Table 4–3 lists the characteristics for 802.3 frame format according to the DLX function and QIO with which you use them. Within each function, the characteristics are listed alphabetically. Each QIO description later in this chapter has detailed information on the related characteristics.

Table 4–3: Characteristics for 802.3 Frame Format

Symbol & Value	What It Does
Open the Port (IO.XOP)	
CC.FMO (103)	Defines a single frame format for the port.
CC.SCO (104)	Requests Class I service for the 802.3 port.
Set and Get Port Characteristics (IO.XSC and IO.XGC)	
CC.FRM (202)	With IO.XSC, enables or disables a second frame format; with IO.XGC, returns the second frame format.
CC.GSP (205)	With IO.XSC, enables or disables the port's processing of a specified group SAP; with IO.XGC, returns information on enabled GSAPs.
CC.ISP (204)	With IO.XSC, enables or disables the port's processing of a specified individual SAP; with IO.XGC, returns information on enabled ISAPs.
CC.MCT (201)	With IO.XSC, enables the port to receive messages with the specified multicast address; with IO.XGC, returns the enabled multicast address.
CC.SNP (206)	With IO.XSC, enables or disables the port's processing of the specified SNAP protocol identifier; with IO.XGC, returns information on enabled SNAP protocols.
CC.SRV (203)	With IO.XSC, requests Class I service; with IO.XGC, returns the value for Class I service.
Transmit and Receive (IO.XTM and IO.XRC)	
CC.ADR (100)	Sets a destination node address on transmit; returns the source node address on receive.
CC.CTM (107)	Specifies the contents of the control field (CTL) on transmit; returns the contents of the control field on receive.
CC.DAD (102)	Returns the destination address of an incoming frame on receive.
CC.FMM (105)	Specifies the frame format of a message on transmit; returns the frame format of a message on receive.
CC.SNM (110)	Specifies the SNAP protocol identifier for a message on transmit; returns the SNAP protocol on receive.
CC.SPM (106)	Specifies the Destination SAP/Source SAP pair for a message on transmit; returns the DSAP/SSAP on receive.

4.4 DLX QIOs

DLX requests conform to normal standards for RSX–11 QIOs, including logical unit numbers (LUNs), event flags, I/O status blocks, asynchronous system traps (ASTs), and parameter lists. According to RSX–11 standards, you can use any one of the three macro formats (see Chapter 2). You can use the QIO wait option (QIOW$) to suspend execution of the program until the call completes.

The rest of this chapter describes the DLX QIOs. The descriptions are in the order in which you will probably use the QIOs. Note that the QIO descriptions include lists of codes for two distinct types of completion status:

- QIO completion status
- Characteristics completion status

QIO completion status codes tell you that the QIO executed successfully or that a specific error occurred during execution. DLX returns the completion status to the 2-word status block that you specify as the *status* parameter in the QIO's format. IS.SUC (1) is the standard success code. The codes for execution errors have an IE. prefix and three letters that represent a specific error. Each QIO description includes any QIO completion status codes you may get.

Characteristics status codes tell you that the characteristics block was successfully processed or that a specific error occurred during processing. DLX returns the status code to the characteristics block's status field, C.STAT. CS.SUC (1) is the code for success. The other codes have a CE. or CS. prefix and three variable letters representing specific status. Each characteristic description lists the status codes you can receive for that characteristic, and Appendix H provides more detailed information on each code.

Note that the QIO can succeed even if the characteristics function encounters an error. For full completion status, check the contents of both the status block and C.STAT field.

IO.XOP
(Open a Port)

4.4.1 IO.XOP — Open a Port

Use:

Issue this QIO to create a port for DLX transmission and reception. The port is an I/O access path to the controller whose device ID you specify in arguments *p4* and *p5*. In response to this QIO, DLX scans the controller's port data base and associates an available port with the logical unit number (LUN) that you specify.

The port will open for Ethernet frame format unless you use the *p4* and *p5* parameters to specify 802.3 format.

Format:

QIO$ IO.XOP,*lun*,[*efn*],,[*status*],[*ast*],<*p1,p2,p3,*[*p4,p5*]>

Arguments:

IO.XOP

is the function code that opens a port.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the QIO completes.

status

is the address of an optional 2-word status block that contains the QIO's completion status in the low-order byte of the first word (see under "QIO Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

p1

is the address of an ASCII string that identifies the controller on which to open the port. The string has the form *dev-ctl*, where *dev* is a device name, such as UNA or QNA, and *ctl* is the decimal value for the controller number.

p2

is the length of an ASCII string that identifies the controller on which to open the port. The string has the form *dev-ctl*, where *dev* is a device name, such as UNA or QNA, and *ctl* is the decimal value for the controller number.

p3

is a word argument that specifies the timeout value and port mode. The timeout value specifies how long to wait to receive a transmitted message. The low-order byte of the word designates the receive timeout value as follows:

timeout = 0 for no receive timer.
timeout = <*n*>

where *n* is the timer value in seconds. (The timer value *n* causes the timeout to have a range of *n*-1 to *n*.)

p4

is the address of the characteristics buffer.

p5 is the length of the characteristics buffer.

Characteristics Buffer:

The characteristics buffer can contain the following blocks:

CC.FMO	Frame Format for Open
CC.SCO	Class I Service for Open

The blocks must be in sequential order in the buffer; that is, the frame format characteristic must precede the service class characteristic.

Refer to Appendix H for more information on the characteristics status codes.

CC.FMO (103) = Frame Format for Open

This characteristic specifies a frame format for opening the port. You can specify one format. To use both Ethernet and 802.3 frame format, specify one when opening the port, and the other when setting port characteristics with IO.XSC.

The C.CHRL field consists of 2 bytes:

- The low byte can contain NX$ETH (1) for Ethernet format or NX$802 (2) for 802.3 format.
- The high byte is reserved.

LKG-1233-87

CC.FMO returns the following status codes in the C.STAT field:

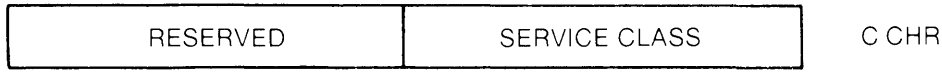
CE.FMI	Illegal frame format.
CE.FMC	Frame usage conflict.
CS.SUC	Success.

CC.SCO (104) = Class I Service for Open

This characteristic requests 802.3 Class I service for your application. You must use CC.SCO to get Class I service; Class II (user-supplied) service is the default.

The C.CHRL field consists of 2 bytes:

- The low byte must contain NX$CLI (4).
- The high byte is reserved.



LKG-1234-87

CC.SCO returns the following characteristics status codes in C.STAT:

CE.FMC	Frame usage conflict.
CE.SRI	Illegal service class.
CS.SUC	Success.

QIO Completion Status:

IS.SUC (1)	The port successfully opened.
177736 IE.ALN (–34.)	The LUN you specified is already in use.
177776 IE.IFC (–2.)	The LUN is not assigned to NX:.
177646 IE.NSF (–26.)	You specified a non-existent line.
177760 IE.PRI (–16.)	The port you specified is not available for use by DLX.
177757 IE.RSU (–17.)	The port you specified is already in use.

IO.XSC
(Set Characteristics)

4.4.2 IO.XSC — Set Characteristics

Use:

Use this QIO to set various port characteristics. The characteristics can include a second frame format, multicast addresses, and protocols.

For a description of the fields in characteristics blocks, refer to Section 4.3.4. Always put a zero (0) in the C.DAT0 field.

Format:

QIO$ IO.XSC,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*>

Arguments:

IO.XSC

is the function code whose parameters specify the location and length of the characteristics buffer.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the call completes.

status

is the address of a 2-word status block that contains completion status. On completion, the second word of the I/O status block indicates how much of the characteristics block was processed.

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

p1

is the address of the characteristics buffer.

p2

is the length of the characteristics buffer. The buffer can contain multiple characteristics blocks.

Characteristics Buffer:

This section has information on the characteristics blocks to use with IO.XSC. The blocks are in alphabetical order.

You can use the following blocks with Ethernet frame format:

CC.DST	Ethernet Protocol Type for Port
CC.FRM	Frame Format for Port
CC.MCT	Multicast Address for Port

You can use the following blocks with 802.3 frame format:

CC.FRM	Frame Format for Port
CC.MCT	Multicast Address for Port
CC.GSP	Group SAP for Port
CC.ISP	Individual SAP for Port
CC.SNP	SNAP Identifier for Port
CC.SRV	Service Class for Port

Enter the characteristics blocks in sequential order in the buffer. For example, specify the frame format before the characteristics that depend on a particular frame format.

Use the data input size field (C.DATI) to indicate how many bytes of data you are supplying.

Refer to Appendix H for more information on characteristics status codes.

CC.DST (200) = Protocol Type for Port

This characteristic contains a protocol type and may include other instructions about the use of the protocol type. It can optionally set protocol flags and specify Ethernet addresses to which to send and from which to receive messages with the protocol type. Unless you set the DF$DIS flag, the characteristic enables the protocol type.

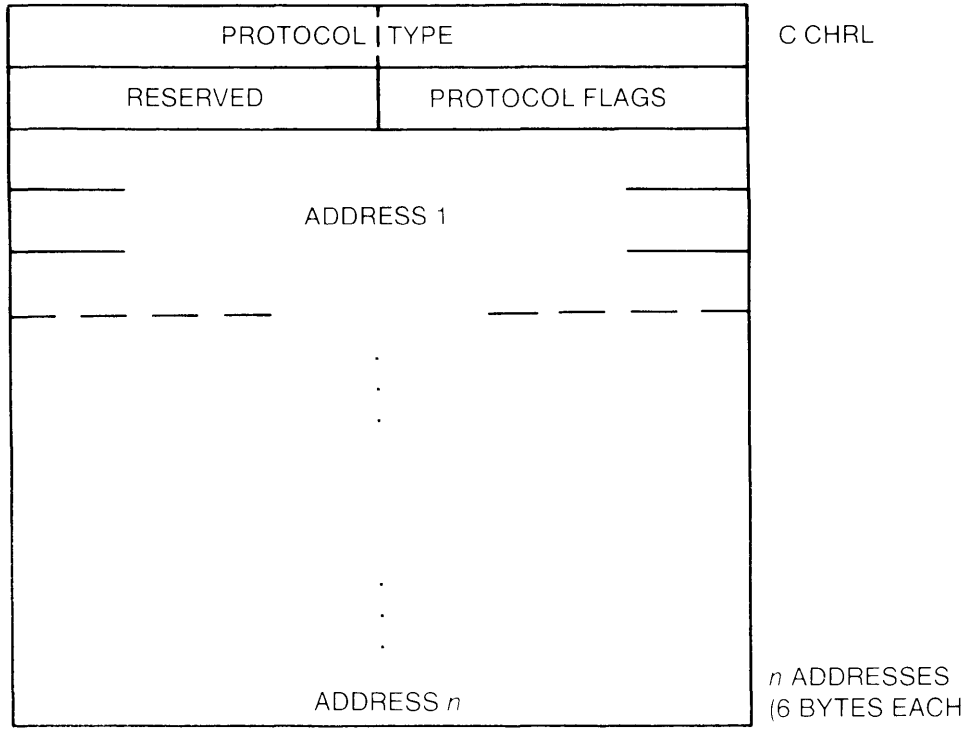
The protocol type must have a value greater than 1500. Store the low byte of the protocol in the high byte of C.CHRL and the high byte of the protocol in the low byte of C.CHRL.

You can use the following protocol flags:

- LF$DIS disables the protocol type.
- LF$EXC causes the port to receive the protocol type in exclusive mode.
- LF$DEF causes the port to receive the protocol type in default mode.
- LF$PAD requests padding support for frames with the protocol type.

You can add the Ethernet addresses of remote nodes to the protocol type characteristic, but not with exclusive or default mode.

The C.DATI field equals $4 + 6n$ bytes, where n is the number of addresses that you include.



LKG-1236-87

CC.DST returns the following characteristics status codes to the C.STAT field:

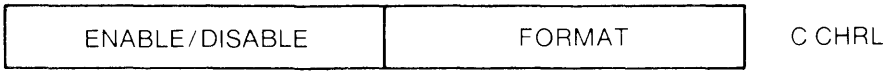
CE.ACN	Address usage conflict.
CE.IUN	Illegal use of multicast address.
CE.PCN	Protocol usage conflict.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDF	Undefined function.
CS.SUC	Success.

CC.FRM (202) = Frame Format for Port

CC.FRM enables or disables a frame format for the port. Use it to enable a format other than the one in which you opened the port.

The C.CHRL field consists of 2 bytes:

- The low byte specifies the format. The low byte can have the value PF$ETH (2) for Ethernet format or PF$802 (4) for 802.3 format.
- The high byte specifies whether to enable or disable the format. A value of zero (0) disables the format; any other value enables it.



LKG-1235-87

CC.FRM returns the following characteristics status codes to the C.STAT field:

CE.FMI	Illegal frame format.
CE.FMC	Frame usage conflict.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDF	Undefined function.
CS.SUC	Success.

CC.GSP (205) = Group SAP for Port

This characteristic enables or disables a specified GSAP for the port. You must first enable Class I service and an Individual Sap. The C.CHRL field consists of two bytes:

- The low byte is the GSAP value, in the range 0 to 255., where bit zero (0) must equal 1.
- The high byte specifies whether to enable or disable. A value of zero (0) disables the GSAP; any other value enables it.

LKG-1237-87

CC.GSP returns the following characteristics status codes to the C.STAT field:

CE.FMC	Frame usage conflict.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDF	Undefined function.
CS.SUC	Success.

CC.ISP (204) = Individual SAP for Port

This characteristic enables or disables a specified Individual SAP. The C.CHRL field consists of two bytes:

- The low byte is the SAP value, in the range 0 to 255., where bit 0 must equal 0.
- The high byte specifies whether to enable or disable. A value of zero (0) disables the ISAP; any other value enables it.

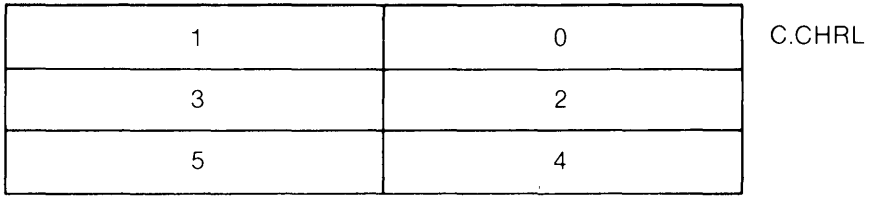
LKG-1238-87

CC.ISP returns the following characteristics status codes to the C.STAT field:

CE.FMC	Frame usage conflict.
CE.SPU	SAP in use.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDG	Undefined function.
CS.SUC	Success.

CC.MCT (201) = Multicast Address for Port

CC.MCT enables the port to receive messages with the specified multicast address. The C.CHRL field consists of 6 bytes.



LKG-1239-87

CC.MCT returns the following characteristics status codes in the C.STAT field:

CE.MCE	Multicast address already enabled.
CE.NMA	Not a multicast address.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDF	Undefined function.
CS.SUC	Success.

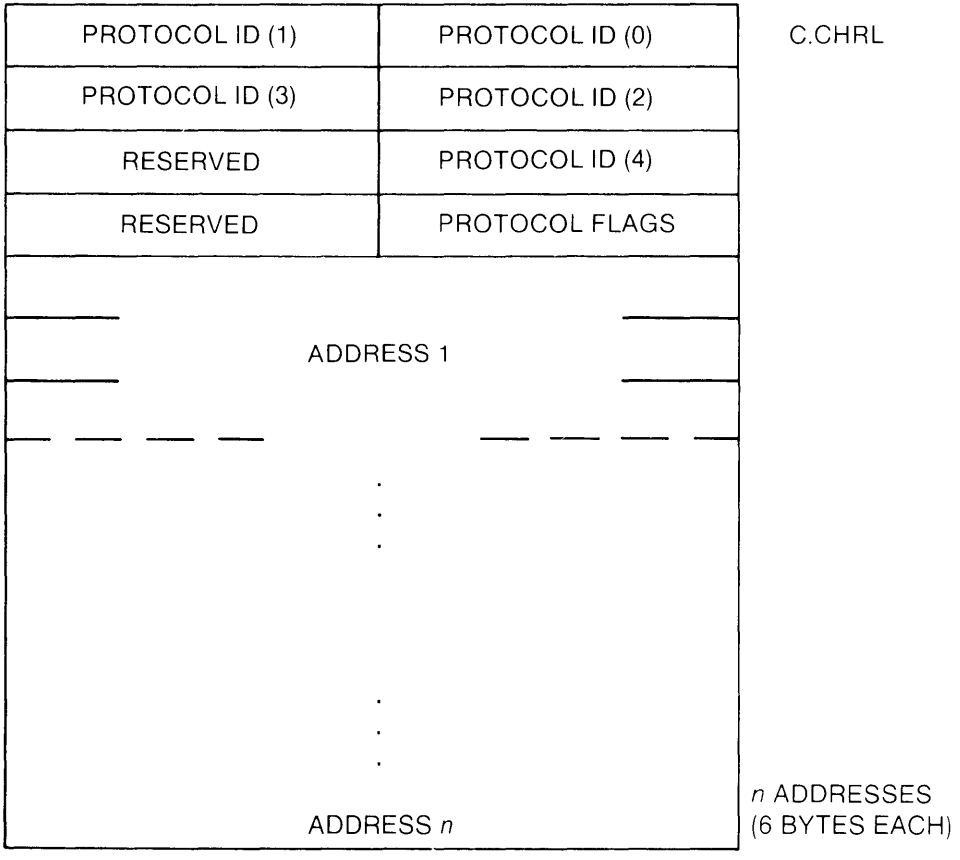
CC.SNP (206) = SNAP Identifier for Port

This characteristic specifies a SNAP identifier and may include other instructions about the use of the protocol. It can optionally set protocol flags and specify Ethernet addresses to which to send and from which to receive messages with the SNAP identifier. Unless you set the LF$DIS flag, the characteristic enables the port to send and receive messages that have the protocol.

The C.CHRL field consists of the following:

- Bytes zero (0) through 4 contain the SNAP identifier.
- Byte 5 is reserved.
- Byte 6 contains any flags to set (LF$EXC for exclusive mode, LF$DEF for default mode, or LF$DIS to disable the SNAP identifier).
- Byte 7 is reserved.
- Successive 6-byte groups contain any Ethernet addresses to pair with the protocol. You cannot add addresses with exclusive or default mode.

The C.DATI field equals 8 + 6 *n* bytes, where *n* is the number of addresses that you include.



LKG-1240-87

CC.SNP returns the following characteristics status codes in the C.STAT field:

CE.FMC	Frame usage conflict.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.SNU	SNAP in use.
CE.UDF	Undefined function.
CS.SUC	Success.

CC.SRV (203) = Service Class for Port

This characteristic specifies 802.3 Class I service. In the buffer, this characteristics block precedes SAP or SNAP identifiers. Use it in conjunction with CC.FRM. C.CHRL has two bytes:

- The low byte specifies the service class and has the value PF$CLI (10).
- The high byte specifies whether to enable or disable. A value of zero (0) disables Class I service; any other value enables it.

LKG-1241-87

CC.SRV returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CE.SRI	Illegal service class.
CE.RES	Resource allocation failure.
CE.RTL	Request too large.
CE.RTS	Request too small.
CE.UDF	Undefined function.
CS.SUC	Success.

IO.XGC
(Get Characteristics)

4.4.3 IO.XGC — Get Characteristics

Use:

Use this QIO to return information on various characteristics of a port. The characteristics to return include the port's frame format, enabled multicast addresses, enabled protocols, and so forth.

For more information on characteristics, refer to section 4.3.4.

Format:

QIO$ IO.XGC,*lun*,[*efn*],,[*status*],[*ast*],<*p1,p2*>

Arguments:

IO.XGC

is the function code whose parameters specify the location and length of a characteristics buffer that returns port characteristics.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the QIO completes.

status

is the address of a 2-word status block that contains the QIO completion status. On completion, the second word of the I/O status block indicates how much of the characteristics block was processed.

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

p1

is the address of the characteristics buffer.

p2

is the length of the characteristics buffer. The buffer can contain multiple characteristics blocks.

Characteristics Buffer:

This section has information on the characteristics blocks to use with IO.XGC. The blocks are arranged alphabetically.

You can use the following blocks with Ethernet frame format:

CC.DST	Ethernet Protocol Type for Port
CC.FRM	Frame Format for Port
CC.MCT	Multicast Address for Port

You can use the following blocks with 802.3 frame format:

CC.FRM	Frame Format for Port
CC.GSP	Group SAP for Port
CC.ISP	Individual SAP for Port
CC.MCT	Multicast Address for Port
CC.SNP	SNAP Identifier for Port
CC.SRV	Class I Service for Port

For characteristics with multiple occurrences, append multiple blocks. Each subsequent block returns the next occurrence of the characteristic. For example, CC.MCT returns an enabled multicast address; for multiple multicast addresses, append multiple CC.MCT blocks in the buffer. If no more occurrences of the characteristic exist, the C.STAT field returns an IE.IGN (ignored) error.

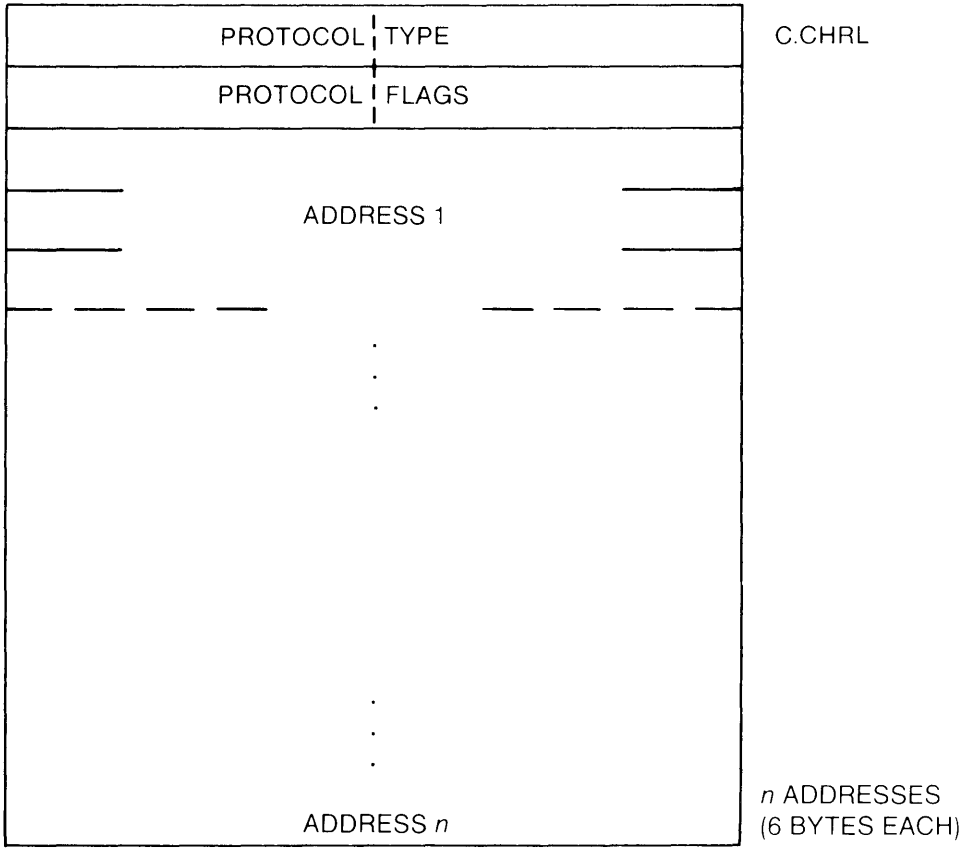
For variable-length fields, such as a protocol that may or may not be paired with a number of addresses, check the contents of the C.DAT0 field for the size of the returned data.

Refer to Appendix H for more information on characteristics status codes.

CC.DST (200) = Ethernet Protocol Type for Port

This characteristic returns information on the enabled Ethernet protocol type, including the protocol flags and any Ethernet addresses you may have paired with the protocol. A recurrence of the characteristics block returns information on the next enabled protocol type.

The protocol type and flags consist of 2 bytes each, and each address consists of 6 bytes. The high byte of C.CHRL returns the low byte of the protocol and the low byte of C.CHRL returns the high byte of the protocol.



LKG-1242-87

The value in the C.DAT0 field equals $4 + 6n$ bytes, where n is the number of addresses for the protocol type.

CC.DST returns the following characteristics status codes in the C.STAT field:

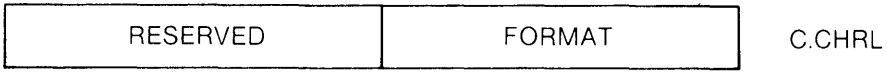
CE.DAO	Data overrun.
CE.RTS	Request too small.
CS.SUC	Success.

CC.FRM (202) = Frame Format for Port

This characteristic returns the current frame format(s) for the port.

The C.CHRL field consists of 2 bytes:

- The low byte returns PF$ETH (2) for Ethernet format, PF$802 (4) for 802.3 format, or PF$ETH!PF$802 (6) for both formats.
- The high byte is reserved.



LKG-1243-87

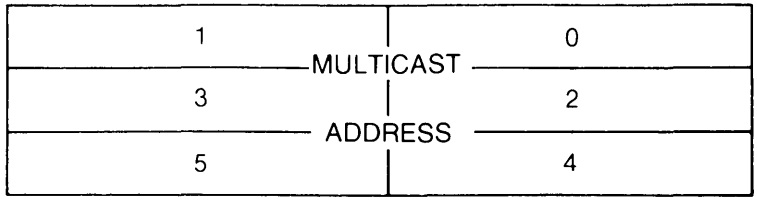
CC.FRM returns the following characteristics status codes in the C.STAT field:

CE.RTL	Request too large.
CE.RTS	Request too small.
CS.SUC	Success.

CC.MCT (201) = Multicast Address for Port

This characteristic returns a multicast Ethernet address currently enabled for the port. A recurrence of the characteristics block returns the next enabled multicast address.

The C. CHRL field consists of 6 bytes. For more information on Ethernet addresses, refer to section 4.3.3.



LKG-1244-87

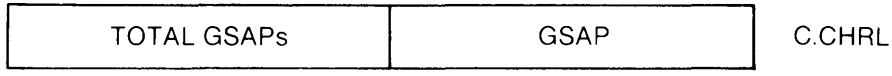
CC.MCT returns the following characteristics status codes in the C.STAT field:

CE.RTL	Request too large.
CE.RTS	Request too small.
CS.SUC	Success.

CC.GSP (205) = Group SAP for Port

This characteristic returns information on enabled Group SAPs. The C.CHRL field has two bytes:

- The high byte returns the number of currently-enabled Group SAPs for the port.
- The low byte returns the first enabled Group SAP for the first occurrence of the block. The next occurrence of the block returns the next enabled Group SAP.



LKG-1245-87

CC.GSP returns the following characteristics status codes in C.STAT:

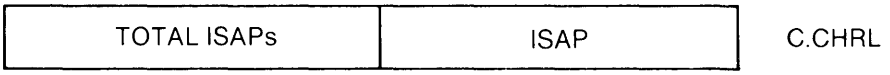
CE.FMC	Frame usage conflict.
CE.RTL	Request too large.
CE.RTS	Request too small.
CS.IGN	Ignored.
CS.SUC	Success.

CC.ISP (204) = Individual SAP for Port

This characteristic returns an enabled Individual SAP. A recurrence of the characteristics block returns the next Individual SAP.

The C.CHRL field consists of two bytes:

- The low byte is the SAP, which is in the range zero (0) to 255, where bit zero (0) equals 0.
- The high byte returns the number of currently-enabled Individual SAPs for the port.



LKG-1246-87

CC.ISP returns the following characteristics status codes in the C.STAT field:

CE.FMC	Frame usage conflict.
CE.RTL	Request too large.
CE.RTS	Request too small.
CS.SUC	Success.

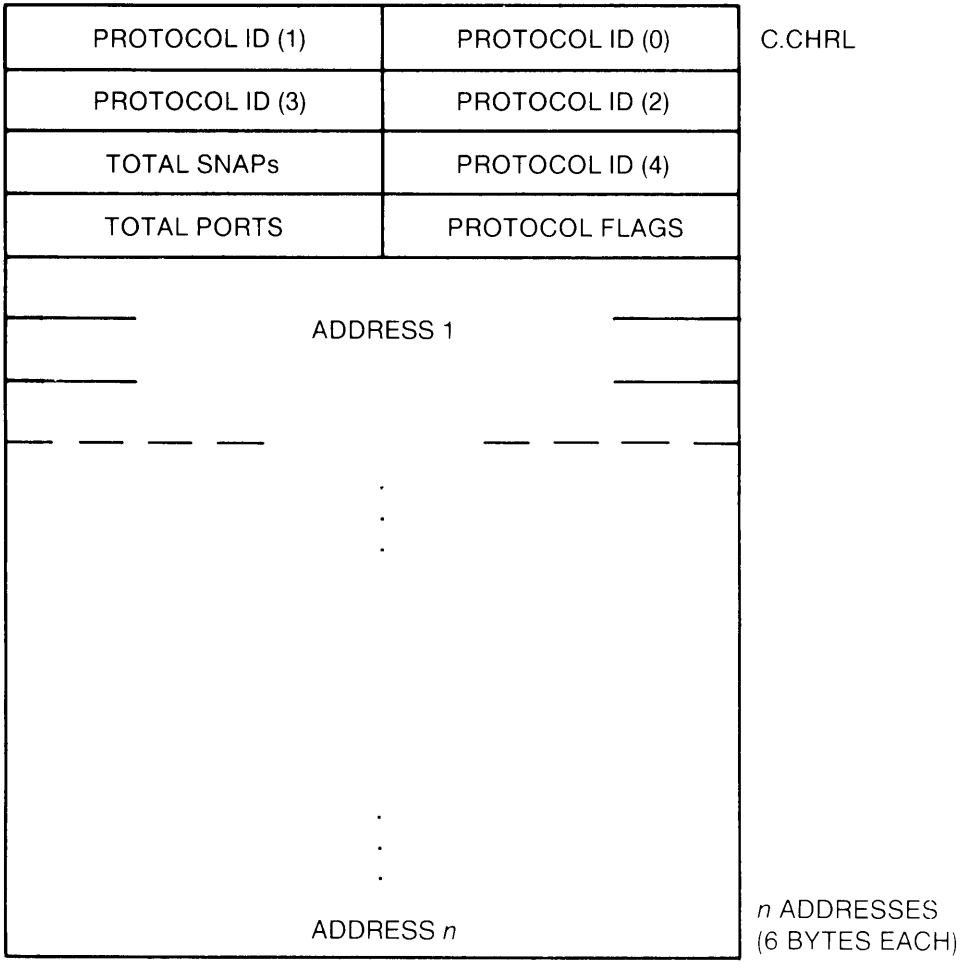
CC.SNP (206) = SNAP Identifier for Port

This characteristic returns information about the SNAP identifiers currently enabled for the port. The first occurrence of the block returns information on the first SNAP; subsequent occurrences of the block each return the next SNAP.

The C.CHRL field consists of 8 bytes:

- Bytes 0–4 contain the SNAP identifier.
- Byte 5 returns the number of currently-enabled SNAP identifiers for the port.
- Byte 6 contains any protocol flags.
- Byte 7 returns the number of ports currently using this SNAP identifier.

- Successive 6-byte groups contain any Ethernet addresses associated with the protocol.



LKG-1247-87

CC.SNP returns the following characteristics status codes in the C.STAT field:

CE.FMU	Frame usage conflict.
CE.RTL	Request too large.
CE.RTS	Request too small.
CS.SUC	Success.

CC.SRV (203) = Class I Service for Port

This characteristic returns the value for 802.3 Class I service when 802.3 is a second frame format. Use it in conjunction with CC.FMM.

The C.CHRL field has two bytes:

- The low byte returns PF$CLI (10) if Class I service is enabled.
- The high byte is reserved.

LKG-1248-87

CC.SRV returns the following characteristics status codes in the C.STAT field:

CE.RTL	Request too large.
CE.RTS	Request too small.
CS.SUC	Success.

IO.XTM
(Transmit a Message on the Port)

4.4.4 IO.XTM — Transmit a Message on the Port

Issue this QIO to transmit a message on an open port. In entering the QIO, you specify the address and length of a buffer that contains the data that you want to transmit. When the QIO executes, it transfers that data to a network buffer.

You can set a number of characteristics for the transmit operation, including the destination address, frame format, protocol, and so forth.

Format:

QIO$ IO.XTM,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*,[*p3*,*p4*]>

Arguments:

IO.XTM

is the function code for transmitting a message.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the call completes.

status

is the address of a 2-word status block that contains completion status. The status block contains the QIO completion status in the low-order byte of the first word (see under "QIO Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after the QIO completes.

p1

is the address of the user buffer that contains the message to transmit.

p2

is the length of the message to transmit.

p3

is the address of the characteristics buffer.

p4

is the length of the characteristics buffer.

Characteristics Buffer:

This section has information on the characteristics blocks to use with IO.XTM. The blocks are described in alphabetical order.

You can use the following characteristics with Ethernet frame format:

CC.ADR	Address for Message
CC.FMM	Frame Format for Message
CC.PRO	Ethernet Protocol Type for Message

You can use the following characteristics with 802.3 frame format:

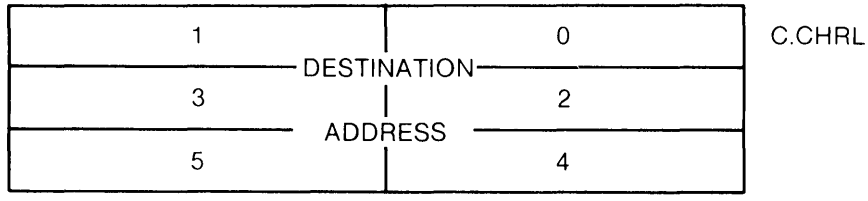
CC.ADR	Address for Message
CC.CTM	Control Field for Message
CC.FMM	Frame Format for Message
CC.SNM	SNAP Identifier for Message
CC.SPM	Destination and Source SAP for Message

Enter the characteristics blocks into the buffer in sequential order. For example, specify the frame format before the characteristics that depend on a particular frame format.

Refer to Appendix H for more information on characteristics status codes.

CC.ADR (100) = Address for Message

This characteristic specifies the Ethernet address to use as a destination for transmission; it is required on each transmission. The address consists of 6 bytes. For information on setting up the Ethernet address for DECnet nodes, refer to the introductory sections of this chapter.



LKG-1249-87

CC.ADR returns the following characteristics status codes in the C.STAT field:

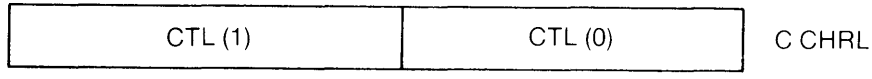
CS.IGN	Ignored.
CS.SUC	Success.

CC.CTM (107) = Control Field for Message

This characteristic specifies the value for the 802.3 control (CTL) field. Use CC.CTM in conjunction with an Individual or Group SAP (ISAP or GSAP) address to specify the contents of the frame.

For user-supplied service, you can create symbolics to use in the control field in accordance with the IEEE standard. For Class I service, you can use the following symbolics:

Symbolic	Message Type
$CSUIF	UI
$CSXIF	XID
$CSTSF	TEST



LKG-1250-87

CC.CTM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.FMM (105) = Frame Format for Message

This characteristic specifies the frame format for the message. The characteristic block is necessary only with 802.3 transmissions, since Ethernet is the default format. The C.CHRL field consists of 2 bytes:

- The low byte contains NX$ETH (1) for Ethernet format or NX$802 (2) for 802.3 format.
- The high byte is reserved.

LKG-1251-87

CC.FMM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.PRO (101) = Protocol Type for Message

This characteristic supplies the message's protocol type. You must supply the protocol type on all Ethernet-format transmissions.

The C.CHRL field contains 2 bytes for the protocol type.

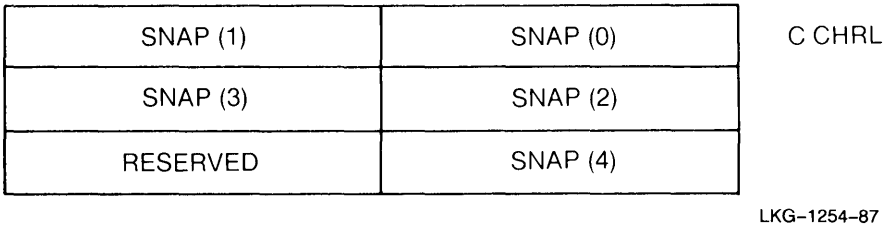
LKG-1252-87

CC.PRO returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.SNM (110) = SNAP Identifier for Message

This characteristic specifies a message's SNAP identifier. Use six bytes, with 5 bytes for the SNAP and 1 byte reserved, as follows:

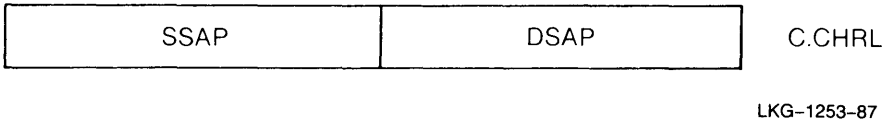


CC.SNM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.SPM (106) = Destination and Source SAPs for Message

This characteristic specifies a message's Destination and Source SAPs. Use it in conjunction with CC.CTM.



CC.SPM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

QIO Completion Status:

IS.SUC (1)	The message was successfully transmitted to the remote node.
177761 IE.ABO (-15.)	The transmission was aborted. Close and reopen the port.
177777 IE.BAD (-1)	You get this code with Ethernet frame format if you omit the protocol type and/or remote address. You get the code with 802.3 frame format if you omit either a SNAP identifier or a DSAP/SSAP pair and control field.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177733 IE.NLN (-37.)	No open port has the specified LUN.
177772 IE.SPC (-6.)	The transmit buffer is too large. This status code applies only to PDP-11/44 or PDP-11/70 with extended memory.

IO.XRC
(Receive a Message on the Port)

4.4.5 IO.XRC — Receive a Message on the Port

Issue this QIO to receive a message from a remote node.

Format:

QIO$ IO.XRC,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*,[*p3*,*p4*]>

Arguments:

IO.XRC

is the function code for receiving a message.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the call completes.

status

is the address of a 2-word status block. The status block contains the QIO completion status in the low-order byte of the first word (see under "QIO Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

p1

is the address of a user buffer to receive the message.

p2

is the length, in bytes, of the user buffer to receive the message. The length of the received message cannot exceed the system buffer, regardless of the length you specify in *p2*.

p3

is the address of the characteristics buffer.

p4

is the length of the characteristics buffer.

Characteristics Buffer:

This section has information on the characteristics blocks to use with IO.XRC. The blocks are in alphabetical order.

You can use the following characteristics with Ethernet frame format:

CC.ADR	Source Address of Message
CC.DAD	Destination Address of Message
CC.FMM	Frame Format of Message
CC.PRO	Ethernet Protocol Type of Message

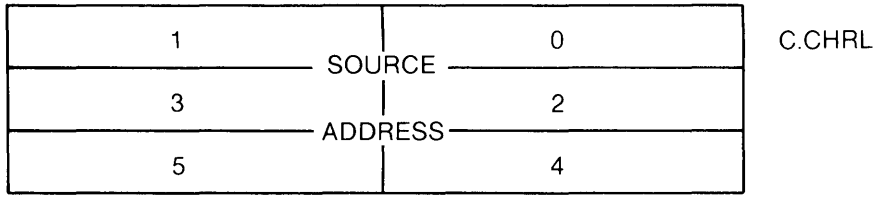
You can use the following characteristics with 802.3 frame format:

CC.ADR	Source Address of Message
CC.CTM	Control Field of Message
CC.DAD	Destination Address of Message
CC.FMM	Frame Format of Message
CC.SNM	SNAP Identifier of Message
CC.SPM	Destination and Source SAPs of Message

Refer to Appendix H for more information on characteristics status codes.

CC.ADR (100) = Address of Message

This characteristic returns the source Ethernet address from which a received frame was sent. The address consists of 6 bytes.



LKG-1255-87

CC.ADR returns the following characteristics status codes in the C.STAT field:

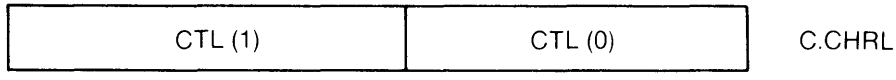
CS.IGN	Ignored.
CS.SUC	Success.

CC.CTM (107) = Control Field of Message

This characteristic returns the 802.3 control (CTL) field's value. The control field tells what type of data the frame contains in frames with DSAP/SSAP identifiers.

In frames with Class I service only, the control field value will be one of the following:

Symbolic	Message Type
$CSUIF	UI
$CSXIF	XID
$CSTSF	TEST



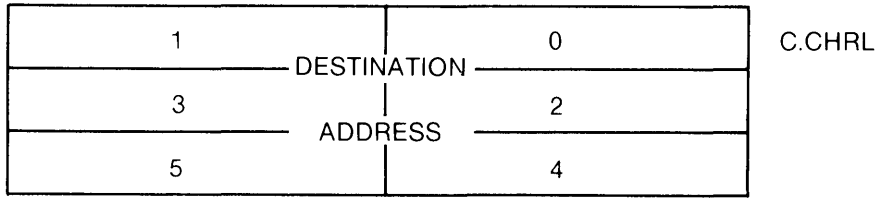
LKG-1256-87

CC.CTM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.DAD (102) = Destination Address of Message

This characteristic returns the destination address on a received frame; it will be either your physical address or one of your enabled multicast addresses. The destination address consists of 6 bytes.



LKG-1257-87

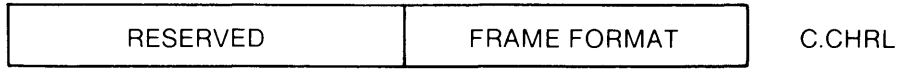
CC.DAD returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.FMM (105) = Frame Format of Message

This characteristic returns the frame format of a received message. The C.CHRL field consists of 2 bytes:

- The low byte returns NX$ETH (1) for Ethernet format or NX$802 (2) for 802.3 format.
- The high byte is reserved.



LKG-1258-87

CC.FMM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.PRO (101) = Protocol Type of Message

CC.PRO returns the protocol type for an Ethernet-format frame. The C.CHRL field contains 2 bytes for the protocol type.

LKG-1252-87

CC.PRO returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.SNM (110) = SNAP Identifier of Message

This characteristic returns the SNAP identifier of a received message. The SNAP identifier has 5 bytes, formatted as follows:

SNAP (1)	SNAP (0)	C.CHRL
SNAP (3)	SNAP (2)	
RESERVED	SNAP (4)	

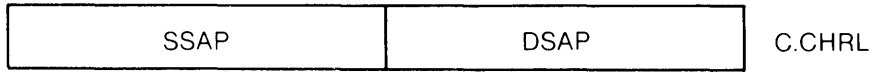
LKG-1259-87

CC.SNM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

CC.SPM (106) = SAPs of Message

CC.SPM returns the Destination SAP (DSAP) and Source SAP (SSAP) address of a received message. The C.CHRL field has two bytes: the low byte stores the DSAP, and the high byte stores the SSAP.



LKG-1260-87

CC.SPM returns the following characteristics status codes in the C.STAT field:

CS.IGN	Ignored.
CS.SUC	Success.

QIO Completion Status:

IS.SUC (1)	You successfully received a message from the remote node. The second word of the I/O status block contains the number of bytes you received.
177761 IE.ABO (-15.)	The receive function was aborted because an unrecoverable error occurred in the hardware device. Close and reopen the port.
177763 IE.DAO (-13.)	Some data was lost because a message arrived before the application issued an IO.XRC directive, or because the user buffer was too small and truncated the message. The user buffer length is in the second word of the I/O status block.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177733 IE.NLN (-37.)	No open port has the specified logical unit number.
177641 IE.TMO (-95.)	A timeout condition occurred. The timer interval that you specified in opening the port expired without a message arriving.

IO.XCL
(Close the Port)

4.4.6 IO.XCL — Close the Port

Use:

Issue IO.XCL to close the port.

Format:

QIO$ IO.XCL,*lun*,[*efn*],,[*status*],[*ast*]

Arguments:

IO.XCL

is the function code that closes the port.

lun

is the logical unit number associated with the port.

efn

is an optional event flag number set when the call completes.

status

is the address of a 2-word status block. The status block contains the QIO completion status in the low-order byte of the first word (see under "QIO Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

QIO Completion Status:

IS.SUC (1)	The port has successfully closed.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177733 IE.NLN (-37.)	No open port has the specified LUN.

4.4.7 DLX QIO Programming Examples

The following programs are examples of programming DLX for an Ethernet. The first example uses 802.3 frame format. The second example uses Ethernet frame format.

4.4.7.1 802.3 Example

This program is a DLX 802.3 test program. You can run the program on two nodes to test the nodes' ability to send and receive 802.3 frames on the data link level. On the transmitting node, the task builds and sends 802.3 XID, TEST, or UI frames. On the receiving node, the task simply returns the received frames to the sender.

```
        .TITLE  802TST - 802.3 Test Tool
        .IDENT  /V01.00/

; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
; This program tests an 802.2 data link.  As a "sender," the
; task builds an 802.3 XID, TEST, or UI command frame and transmits it
; to the remote responder.  As a "receiver," the task transmits the
; received frames back to the sender.  The sending task then logs the
; returned data to the terminal or to a file.
;
; The program is initiated by command line options:
;
;       Option                  Default         Comment
;       ------                  -------         -------
;
;       /DEV[ICE]=ddd-n         /DEVICE=UNA-0   Controller to use
;       /DSA[P]=n               /DSAP=4         Destination SAP
;       /GSA[P]=n               none            Group SAP
;       /HAR[DWARE]=nn-...-nn   none            Remote hardware address
;       /ISA[P]=n               /ISAP=4         Individual SAP
;       /LOG                    none            Log to 802TST.DAT
;       /MES[SAGE]=message      /MES=XID        Build XID, TEST or UI
;       /NOD[E]=n.m             none            Remote DECnet address
;       /PHY[SICAL]=nn-...-nn   none            Remote physical address
;       /RES[PONDER]            none            Responder mode
;       /SIZ[E]=n               60.             Size to transmit
;       /USE[R]                 none            User-supplied service
;       /SNA[P]=nn-...-nn       none            SNAP protocol identifier
;
; The following commands configure 802TST to respond to frames sent either
; to SAP 8 or to SNAP protocol identifier 01-02-03-04-05.  The frames will
; come from the controller QNA-0 on node 55.202:
;
;       >INS 802TST
;       >802 /DEV=QNA-0/RES/ISAP=8/SNAP=01-02-03-04-05
```

```
;
; The program loops back received 802.3 frames until it is aborted.
;
; The following commands are executed on a different system on the same LAN.
; They cause 802TST to send a 60. byte TEST message to the above image of
; 802TST on node 55.202.  They also cause the program to log the response to
; the file SY:[current]802TST.DAT:
;
;       >INS 802TST
;       >802 /DEV=QNA-0/ISAP=16/DSAP=8/NOD=55.202/MES=TEST/LOG
;
; The program exits after receiving and logging the response. Note that
; the responder program must be started *before* the sender program transmits
; any frames.
;
; To assemble, use the following command string:
;
;       MAC 802TST,802TST/-SP/LI:TTM =IN:[130,10]NETLIB/ML,[200,200]802TST
;
; To task build, use the following command string:
;
;       802TST/PR:0, 802TST/-SP =
;       802TST
;       [101,124]NETLIB/LB
;       /
;       UNITS=4
;       ASG=TI:1:2:3:4
;       TASK=...802
;       GBLDEF=$HELP:0
;       //
;
;    Note: the IN: device must be the DECnet distribution device
;          after the PREGEN (if any) has been performed.
;-

        .SBTTL  Macros

        .MCALL  DIR$,ALUN$S,QIOW$,QIO$,CLEF$S,SETF$S,WTSE$S,EXIT$S,EXST$S
        .MCALL  GTIM$S,SREX$S,ENAR$S,DSAR$S,ASTX$S
        .MCALL  GCMLB$,GCML$,ISTAT$,STATE$,TRAN$
        .MCALL  FCSMC$
        .MCALL  DLXDF$,CHRDF$,CSMDF$

        FCSMC$                          ; Define FCS macros
        DLXDF$  ,,,ETHERNET             ; Define DLX and EPM symbols
        CHRDF$                          ; Define characteristics
        CSMDF$                          ; Define CSMA/CD symbols

        .MACRO  SAVRG   list
        .IRP    reg,<list>
        MOV     reg,-(SP)
        .ENDM
        .ENDM   SAVRG

        .MACRO  RESRG   list
```

(continued on next page)

```
        .IRP    reg,<list>
        MOV     (SP)+,reg
        .ENDM
        .ENDM   RESRG

        .MACRO  TYPE    adr,len,vfc
        MOV     adr,OUTQIO+Q.IOPL
        MOV     len,OUTQIO+Q.IOPL+2
        .IF NB,vfc
        MOV     vfc,OUTQIO+Q.IOPL+4
        .IFF
        MOV     #40,OUTQIO+Q.IOPL+4
        .ENDC
        DIR$    #OUTQIO
        .ENDM   TYPE

        .MACRO  LOG     adr,len,?A,?B
        BIT     #OP.LOG,OPTFLG
        BNE     A
        TYPE    adr,len
        BR      B
A:      PUT$    #LOGFIL,adr,len
B:
        .ENDM   LOG

        .MACRO  FORMAT  adr,fmt
        MOV     adr,R2
        MOV     fmt,R1
        MOV     #FMTBUF,R0
        CALL    $EDMSG
        MOV     R1,FMTL
        .ENDM   FORMAT

        .MACRO  ERROR   adr,fmt
        FORMAT  <adr>,<fmt>
        LOG     #FMTBUF,FMTL
        .ENDM   ERROR

        .MACRO  CHRGEN  code,size
        .WORD   code            ; Characteristic type = code
        .WORD   size            ; Buffersize = size bytes
        .WORD   0,0             ; Reserved, status
        .ENDM   CHRGEN

        .MACRO  OFFSET  block,symbol
        .IIF    EQ,<BF-1>       symbol  = .-block'1
        .ENDM   OFFSET

        .SBTTL  Local constants

EQUALS  = '=
SPACE   = 40
TAB     = 11

CMDLUN  = 1
```

```
TILUN   = 2
CHNLUN  = 3
LOGLUN  = 4
CMDEFN  = 1
TIEFN   = 2
CHNEFN  = 3
DONE    = 4

ISAP    = 4                             ; Default Individual/Source SAP
DSAP    = 0                             ; Default Destination SAP
AREABT  = 176000                        ; Area mask
MXAREA  = 63.                           ; Area limit
MXNODE  = 1023.                         ; Node limit
MXCTL   = 377                           ; Maximum control value
XID1    = ^B10000001                    ; XID message bytes
XID2    = ^B00000001                    ;
XID3    = ^B00000000                    ;
MAXFRM  = 1492.                         ; Maximum frame data
BUFSIZ  = MAXFRM                        ; Receive buffer length
NUMBUF  = 4.                            ; Number of receive buffers
TMO     = 5                             ; Time-out value in seconds

        .SBTTL  Impure data
        .PSECT  $IDATA  D,RW
        .SBTTL  .       .TPARS action routine state variables
OPTFLG: .BLKW   1                       ; Option flags
  OP.DEV = 1                            ;   /DEV[ICE]
  OP.DSP = 2                            ;   /DSA[P]
  OP.GSP = 4                            ;   /GSA[P]
  OP.HDW = 10                           ;   /HAR[DWARE_ADDRESS]
  OP.ISP = 20                           ;   /ISA[P]
  OP.LOG = 40                           ;   /LOG
  OP.MSG = 100                          ;   /MES[SAGE_TYPE]
  OP.NOD = 200                          ;   /NOD[E]
  OP.PHY = 400                          ;   /PHY[SICAL_ADDRESS]
  OP.RSP = 1000                         ;   /RES[PONDER]
  OP.SIZ = 2000                         ;   /SIZ[e]
  OP.USR = 4000                         ;   /USE[R]
  OP.SNP = 10000                        ;   /SNA[P]
MSGFLG: .BLKW   1                       ; Message type flags
  OP.TST = 1                            ;       TEST
  OP.UIF = 2                            ;       UI
  OP.XID = 4                            ;       XID

.MSADR: .BLKW   1                       ; Message address
.MSLEN: .BLKW   1                       ;         length
.NDADR:                                 ; Remote node address (48 bits)
.HWADR:                                 ; Hardware address
.PHADR: .BLKW   3                       ; Physical address
.CTL:   .BLKW   1                       ; CTL field
.SIZE:  .BLKW   1                       ; Size of data block to transmit
.DEVNM: .BLKW   2                       ; Device length, address
.NODID: .BLKW   2                       ; Node address (area, node)
.HXADR: .BLKW   3                       ; Hex address
.GSAP:  .BLKB   1                       ; Group SAP
```

(continued on next page)

```
.ISAP:  .BLKB   1                       ; Individual SAP
.DSAP:  .BLKB   1                       ; Destination SAP
.SNAP:  .BLKB   5                       ; SNAP protocol identifier
.HXDIG: .BLKB   1                       ; Hex digit
.HXBYT: .BLKB   1                       ; Hex byte
        .EVEN

        .SBTTL  .       Directive parameter blocks
OUTQIO: QIOW$   IO.WVB,TILUN,TIEFN,,,,<0,0,40>
OPNQIO: QIOW$   IO.XOP,CHNLUN,CHNEFN,,CHNSB,,<0,0,0,OPNCHB,0>
SETQIO: QIOW$   IO.XSC,CHNLUN,CHNEFN,,CHNSB,,<0,0>
RCVQIO: QIO$    IO.XRC,CHNLUN,,,,RCVAST,<0,BUFSIZ,0,RC.LEN>
XMTQIO: QIOW$   IO.XTM,CHNLUN,CHNEFN,,CHNSB,,<0,0,0,0>
CLSQIO: QIOW$   IO.XCL,CHNLUN,CHNEFN

        .SBTTL  .       Log file structures
LOGFIL: FDBDF$
        FDAT$A  R.VAR,FD.CR
        FDRC$A  ,FMTBUF,132.
        FDOP$A  LOGLUN,,LOGDFN
        FSRSZ$  2                       ; Allocate space for 2 files

        .SBTTL  .       Channel characteristics buffers
;
; Open characteristics buffer
;
OPNCHB:
;
;       Define frame format (=802.3)
;
        CHRGEN  CC.FMO,2
        .WORD   NX$802                  ; Frame format is 802.3
OPNCL0  = . - OPNCHB                    ; Length for user service
;
;       Define service class (Class I)
;
        CHRGEN  CC.SCO,2
        .WORD   NX$CLI                  ; Service class is Class I
OPNCHL  = . - OPNCHB                    ; Length for Class I service

;
; Set characteristics buffers
;
SETBF1:
;
;       Define Individual SAP
;
        CHRGEN  CC.ISP,2
SETISP: .BLKB   1                       ; Individual SAP
        .BYTE   1                       ; Enable flag (0=disable)
SETLN1  = .-SETBF1
;
;       Define Group SAP

 SETBF1:
```

```
        CHRGEN  CC.GSP,2
SETGSP: .BLKB   1                       ; Group SAP
        .BYTE   1                       ; Enable flag
SETLN2  = .-SETBF2
;
;       Define SNAP protocol identifier
;
SETBF3:
        CHRGEN  CC.SNP,8.
SETSNA: .BLKB   5                       ; SNAP protocol identifier
        .BLKB   1                       ; RESERVED
        .BYTE   LF$EXC                  ; Exclusive use
        .BLKB   1                       ; RESERVED
SETLN3  = .-SETBF3

        .SBTTL  .       Message characteristics buffers
;
; Transmit characteristics buffer
;
XMTBFR:     .
;
;       Define Ethernet address
;
        CHRGEN  CC.ADR,6
XMTADR: .BLKW   3                       ; Ethernet address
;
;       Define frame format (=802.3)
;
        CHRGEN  CC.FMM,2
        .WORD   NX$802                  ; 802.3 frame format
;
;       Define Destination SAP and Source SAP
;
        CHRGEN  CC.SPM,2
XMTDSP: .BLKB   1                       ; Destination SAP
XMTSSP: .BLKB   1                       ; Source SAP
;
;       Define PDU type
;
        CHRGEN  CC.CTM,2
XMTCTL: .WORD   0                       ; 802.3 message type
XMTLN1  = .-XMTBFR
;
;       Define SNAP protocol identifier
;
        CHRGEN  CC.SNM,6
XMTSNA: .BLKW   3
XMTLN2  = .-XMTBFR

        .SBTTL          Ring buffers

        BF      = 0
        .REPT   NUMBUF
        BF      = BF+1
        .IRP    N,<\BF>
```

(continued on next page)

```
;
; Buffer descriptor block #'n
;
        .PSECT  $IBDB   D,RW
BDB'n:
                        ; I/O status block
        OFFSET  BDB,BD.STS
        .BLKW   2
                        ; Link to next BDB in ring
        OFFSET  BDB,BD.LNK
        .IF     LT,<BF-NUMBUF>
        .IRP    NEXT,<\BF+1>
        .WORD   BDB'NEXT
        .ENDM
        .IFF
        .WORD   BDB1
        .ENDC
                        ; Data buffer address
        OFFSET  BDB,BD.BUF
        .WORD   BUF'N
                        ; Received chr buffer address
        OFFSET  BDB,BD.RCH
        .WORD   RCH'N
                        ; Transmitted chr buffer address
        OFFSET  BDB,BD.XCH
        .WORD   XCH'N
                        ; Buffer Descriptor Block length
        OFFSET  BDB,BD.LEN

;
; Data buffer #'n
;
        .PSECT  $IBUF   D,RW
BUF'n:
        .BLKB   BUFSIZ
        .EVEN

;
; Received characteristics buffer #'n
;
        .PSECT  $IRCH   D,RW
RCH'n:
                        ; Destination Ethernet address
        CHRGEN  CC.DAD,6
        OFFSET  RCH,RC.DAD
        .BLKW   3
                        ; Source Ethernet address
        CHRGEN  CC.ADR,6
        OFFSET  RCH,RC.SAD
        .BLKW   3
                        ; SNAP protocol identifier
        CHRGEN  CC.SNM,6
        OFFSET  RCH,RC.SNM
        .BLKW   3
                        ; Destination and source SAPs
```

```
        CHRGEN  CC.SPM,2
        OFFSET  RCH,RC.SPM
        .BLKW   1
                        ; Control field
        CHRGEN  CC.CTM,2
        OFFSET  RCH,RC.CTM
        .BLKW   1
                        ; Received chr buffer length
        OFFSET  RCH,RC.LEN

;
; Transmitted characteristics buffer #'n
;
        .PSECT  $IXCH   D,RW
XCH'n:
                        ; Frame format (802.3)
        CHRGEN  CC.FMM,2
        .WORD   NX$802
                        ; Destination Ethernet address
        CHRGEN  CC.ADR,6
        OFFSET  XCH,XC.ADR
        .BLKW   3
                        ; Destination and Source SAPs
        CHRGEN  CC.SPM,2
        OFFSET  XCH,XC.SPM
        .BLKW   1
                        ; Control field
        CHRGEN  CC.CTM,2
        OFFSET  XCH,XC.CTM
        .BLKW   1
                        ; Transmit chr buffer length (DSAP/SSAP/CTL)
        OFFSET  XCH,XC.LN1
                        ; SNAP protocol identifier
        CHRGEN  CC.SNM,6
        OFFSET  XCH,XC.SNM
        .BLKW   3
                        ; Transmit chr buffer length (SNAP protocol)
        OFFSET  XCH,XC.LN2

        .ENDM
        .ENDR

        .SBTTL  .       Miscellaneous local storage
CMDBUF: .BLKB   134.                    ; Command buffer
NUMRCV: .BLKW   1                       ; Number of received frames
EXSTAT: .BLKW   1                       ; Exit status
FMTDAT: .BLKW   25.                     ; Data buffer for formatting

RCVSB:  .BLKW   2                       ; Receive status
CHNSB:  .BLKW   2                       ; Channel status
IOSB:   .BLKW   1                       ; Address of receive status block
FMTL:   .BLKW   1                       ; Length of formatted record
TIMBUF: .BLKW   8.                      ; Time buffer
TIMOUT: .BLKW   1                       ; Time-out for receives
FMTBUF: .BLKW   300.
```

(continued on next page)

```
        .SBTTL  Pure data
        .PSECT  $PDATA  D,RO

        .SBTTL  .       Text strings
        .NLIST  BEX
ERRFM1: .ASCIZ  \%N802 -- %I, $DSW is %D.\
ERRFM2: .ASCIZ  \%N802 -- %I, I/O status is %P %P\
NSFFMT: .ASCIZ  \No such file\
ASNFM1: .ASCIZ  \Cannot assign LUN to channel\
ASNFM2: .ASCIZ  \Cannot assign LUN to command terminal\
ASNFM3: .ASCIZ  \Cannot assign LUN to output terminal\
OPNFMT: .ASCIZ  \Cannot open line\
SETFMT: .ASCIZ  \Error defining ISAP\
RCVFMT: .ASCIZ  \Receive error\
XMTFMT: .ASCIZ  \Transmit error\
DEVDFB: .ASCII  \UNA-0\
DEVDFL  =.-DEVDFB
GCLERR: .ASCIZ  \%NGet command line error, code = %P\
PRSERR: .ASCII  \%NSyntax error: "%VA"\
        .ASCII  \%N%4SOPTFLG %P  MSGFLG %P\
        .ASCII  \%N%4S.DEVNM "%VA"  .NDADR %P %P %P\
        .ASCII  \%N%4S.ISAP %D.  .GSAP %D.  .DSAP %D.  .CTL %D.\
        .ASCIZ  \%N%4S.SNAP %5B\
TIMFMT: .ASCIZ  /%N%Y %3Z/

OPNCHF: .ASCII  \%NOpen characteristics:\
        .ASCII  \%NCC.FMO (=%P) %P %P %P   %P\
        .ASCIZ  \%NCC.SCO (=%P) %P %P %P   %P\

SETCHF: .ASCII  \%NSet characteristics:\
        .ASCII  \%NCC.ISP (=%P) %P %P %P   %P\
        .ASCII  \%NCC.GSP (=%P) %P %P %P   %P\
        .ASCIZ  \%NCC.SNP (=%P) %P %P %P   %P %P %P\
        .NLIST  BEX

XMTCHF: .ASCII  \%NTransmit characteristics:\
        .ASCII  \%NCC.ADR (=%P) %P %P %P   %P %P %P\
        .ASCII  \%NCC.FMM (=%P) %P %P %P   %P\
        .ASCII  \%NCC.SPM (=%P) %P %P %P   %P\
        .ASCII  \%NCC.CTM (=%P) %P %P %P   %P\
        .ASCIZ  \%NCC.SNM (=%P) %P %P %P   %P %P %P\

RCVMS1: .ASCII  <12><15>/Destination Ethernet Address:/
RCVLN1  = .-RCVMS1
RCVMS2: .ASCII  /Source Ethernet Address:/
RCVLN2  = .-RCVMS2
RCVMS3: .ASCII  /Destination SAP, Source SAP and CTL bytes:/
RCVLN3  = . - RCVMS3
RCVMS4: .ASCII  /SNAP protocol identifier:/
RCVLN4  = . - RCVMS4
RCVMS5: .ASCIZ  /Received data, %D. bytes:/
        .EVEN

        .SBTTL          Canned XID, UI and TEST messages
```

```
;
; XID message
;
XIDMSG: .BYTE   XID1,XID2,XID3
XIDLEN  = .-XIDMSG
;
; UI and TEST messages
;
; for (i=0; i<MAXFRM; i++)
;       (buf(i) = i%256)
;
UIFMSG:
TSTMSG:
        $$$1 = 0
        .REPT   MAXFRM
        $$$2 = $$$1/256.
        $$$3 = $$$1-<$$$2*256.>         ; $$$3 = $$$1 mod 256.
        .BYTE   $$$3
        $$$1 = $$$1+1
        .ENDR
        .EVEN

        .SBTTL  .       Miscellaneous pure data
BDBLST: .WORD   BDB1                    ; Address of first buffer descriptor
LOGDFN: NMBLK$  802TST,DAT,,SY,0        ; Log file default filename block
GCLBLK: GCMLB$  2,802,CMDBUF,CMDLUN,,132. ; Get command line block

        .SBTTL  TST802 - Mainline code
        .PSECT  $CODE   I,RO

TST802: FINIT$
        MOV     #EX$SUC,EXSTAT          ; Assume success
        CALL    ASNLNS                  ; Assign channel, command and error LUNs
        BCC     NXTCMD                  ; If CC, proceed
        JMP     EXIT
NXTCMD:
        CLR     OPTFLG                  ; Zero options flag
        CLR     MSGFLG                  ; Zero message type flag
        MOV     #DEVDFL,.DEVNM          ; Set default device string
        MOV     #DEVDFB,.DEVNM+2        ;   ("UNA-0")
        MOVB    #ISAP,.ISAP             ; Set default individual SAP (4)
        CLRB    .DSAP                   ; Set default destination SAP (NULL)
        MOV     #60.,.SIZE              ; Set default data block size (60)
        MOVB    #$CSXIF,.CTL            ; Set default message type (XID)
        MOV     #XIDMSG,.MSADR          ; Set default message address
        MOV     #XIDLEN,.MSLEN          ;                       and length

        CLEF$S  #DONE                   ; Clear exit flag

        GCML$   #GCLBLK                 ; Retrieve a command line
        MOVB    G.ERR(R0),R5            ; Pick up error byte
        BCC     20$                     ; If CC, we're ready to parse

        CMPB    R5,#GE.EOF              ; End_of_file?
        BNE     10$                     ; If NE, no
```

(continued on next page)

```
        JMP     EXIT                    ; EOF - just exit

10$:    FORMAT  R5,#GCLERR              ; Format error status
        LOG     #FMTBUF,FMTL            ;   and report the error
        JMP     EXIT                    ; Exit

20$:    MOV     #3*256.,R1              ; Abbreviate to three characters
        MOV     #KEYTBL,R2              ; Get address of key table
        MOV     G.CMLD(R0),R3           ; Get command line length
        BEQ     NXTCMD                  ; If EQ, get another command line
        MOV     G.CMLD+2(R0),R4         ; Get command line address
        MOV     #START,R5               ; Get address of first state
        CALL    .TPARS                  ; Parse the command line
        BCC     DOCMD                   ; If CC, ok - proceed

        CALL    PRSDMP
        JMP     NXTCMD

DOCMD:
        CALL    OPNLOG                  ; Open log file, if /LOG
                                        ; (Ignore possible open error)

        CALL    OPNCHN                  ; Open an Ethernet channel
        BCS     EXIT                    ;   - exit if error

        CALL    SETCHN                  ; Set the channel characteristics
        BCS     EXIT                    ;   - exit if error
;
; Fill the ring with receives
;
        DSAR$S                          ; Disable AST recognition
        MOV     BDBLST,R0               ; Get first buffer descriptor addr
        MOV     #NUMBUF,R1              ; Get number of buffers in ring
        CLR     NUMRCV                  ; Zero received frame count

10$:    MOV     R0,RCVQIO+Q.IOSB          ; Set IOSB address
        MOV     BD.BUF(R0),RCVQIO+Q.IOPL  ; Set data buffer address
        MOV     BD.RCH(R0),RCVQIO+Q.IOPL+4 ; Set receive char address
        DIR$    #RCVQIO                 ; Post receive
        BCS     EXIT                    ; If CS, error - all done
        SOB     R1,10$                  ; Loop through the ring

        CLEF$S  #DONE

        BIT     #OP.RSP,OPTFLG          ; Are we a passive responder?
        BNE     20$                     ; If NE, yes - don't send command
;
; Transmit an 802.3 command PDU
;
        MOV     .PHADR,XMTADR           ; Set Ethernet address for transmit
        MOV     .PHADR+2,XMTADR+2       ;
        MOV     .PHADR+4,XMTADR+4       ;
        MOVB    .DSAP,XMTDSP            ; Set DSAP for transmit
        MOVB    .ISAP,XMTSSP            ; Set SSAP for transmit
        MOVB    .CTL,XMTCTL             ; Set CTL field for transmit
```

```
        MOV     .MSADR,XMTQIO+Q.IOPL    ; Set adr of message to XMT
        MOV     .MSLEN,XMTQIO+Q.IOPL+2  ; Set len of message to XMT
        MOV     #XMTBFR,XMTQIO+Q.IOPL+4 ; Set adr of XMT char buffer
        MOV     #XMTLN1,XMTQIO+Q.IOPL+6 ; Set len of XMT char buffer
                                        ; (Assume using only DSAP/SSAP/CTL)

        BIT     #OP.SNP,OPTFLG          ; Use only DSAP/SSAP/CTL for XMT?
        BEQ     15$                     ; If EQ, yes - no SNAP specified

        MOV     #.SNAP,R0               ; Get address of stored SNAP
        MOV     #XMTSNA,R1              ; Get address of SNAP buffer
        MOV     #5,R2                   ; Set number of bytes in protocol
12$:    MOVB    (R0)+,(R1)+             ; Move a protocol byte into buffer
        SOB     R2,12$                  ; Loop until done
        MOV     #XMTLN2,XMTQIO+Q.IOPL+6 ; Set len of transmit char buffer to
                                        ;   include SNAP protocol ident

15$:    DIR$    #XMTQIO                 ; Transmit a command PDU
        BCS     EXIT                    ; If CS, directive error
        TSTB    CHNSB                   ; Get an I/O error?
        BMI     EXIT                    ; If MI, yes

20$:    ENAR$S                          ; Enable AST recognition
        SREX$S  #ABOAST                 ; Specify abort AST
        WTSE$S  #DONE                   ; Wait for receive(s)

        DSAR$S                          ; Disable AST recognition
        CLOSE$  #LOGFIL                 ; Close the log file
        DIR$    #CLSQIO                 ; Close the Ethernet channel
        JMP     NXTCMD                  ;   and process the next command

XMTER1:
XMTER2:

EXIT:   DIR$    #CLSQIO                 ; Close the Ethernet channel
        CLOSE$  #LOGFIL                 ; Close the log file
        EXST$S  EXSTAT                  ; Try to exit-with-status
        EXIT$S                          ; Else, just exit

        .SBTTL  AST routines
        .SBTTL  .       RCVAST - channel read complete
;+
; **-RCVAST - AST for channel read complete
;
; Inputs:
;       (SP) = address of I/O status block
;
; Outputs:
;       Message read from channel is formatted and logged
;-

RCVAST:
        MOV     (SP),IOSB               ; Save BDB/status block address
        MOV     R3,(SP)                 ; Save R3-R5
        SAVRG   <R4,R5>                 ;
```

(continued on next page)

```
        MOV     IOSB,R3                 ; Retrieve BDB/status block address
        TSTB    (R3)                    ; Receive error?
        BMI     90$                     ; If MI, yes

        INC     NUMRCV                  ; Count one more buffer filled
        MOV     BD.RCH(R3),R4           ; Get rcved chr buffer address
        MOV     BD.XCH(R3),R5           ; Get xmted chr buffer address

        BIT     #OP.RSP,OPTFLG          ; Are we a passive responder?
        BEQ     20$                     ; If EQ, no - don't send response

        MOV     R3,XMTQIO+Q.IOSB            ; Set IOSB address
        MOV     BD.BUF(R3),XMTQIO+Q.IOPL    ; Set data buffer address
        MOV     BD.STS+2(R3),XMTQIO+Q.IOPL+2 ;                 and length
        MOV     BD.XCH(R3),XMTQIO+Q.IOPL+4  ; Set xmt char address
        MOV     #XC.LN1,XMTQIO+Q.IOPL+6     ;              and length
                                            ; (Assuming DSAP/SSAP/CTL)

        BIT     #OP.SNP,OPTFLG          ; Use only DSAP/SSAP/CTL for XMT?
        BEQ.    15$                     ; If EQ, yes - no SNAP specified

        SAVRG   <R0,R1,R2>
        MOV     R4,R0                   ; Form pointer to received SNAP
        ADD     #RC.SNM,R0              ;
        MOV     R5,R1                   ; Form pointer to transmitted SNAP
        ADD     #XC.SNM,R1              ;
        MOV     #5,R2                   ; Set size of SNAP protocol ident
12$:    MOVB    (R0)+,(R1)+             ; Copy a protocol byte
        SOB     R2,12$                  ; Loop until done
        RESRG   <R2,R1,R0>
        MOV     #XC.LN2,XMTQIO+Q.IOPL+6 ; Include SNAP in XMT char buffer
                                        ; (Note: SNAP supersedes DSAP/...
        BR      17$                     ; Join common code

15$:    MOV     RC.SPM(R4),XC.SPM(R5)   ; Get the DSAP/SSAP from receive
        SWAB    XC.SPM(R5)              ;   and swap them for response
        MOV     RC.CTM(R4),XC.CTM(R5)   ; Set the CTL field

17$:    MOV     RC.SAD(R4),  XC.ADR(R5)  ; Set Ethernet address for transmit
        MOV     RC.SAD+2(R4),XC.ADR+2(R5);
        MOV     RC.SAD+4(R4),XC.ADR+4(R5);
        DIR$    #XMTQIO                 ; Send response PDU
        BCS     90$                     ; If CS, directive error
        BR      30$                     ; Don't log data while responding

20$:    CALL    LOGRCV                  ; Log received data

30$:    MOV     R3,RCVQIO+Q.IOSB            ; Set IOSB address
        MOV     BD.BUF(R3),RCVQIO+Q.IOPL    ; Set data buffer address
        MOV     #BUFSIZ,RCVQIO+Q.IOPL+2     ;                 and length
        MOV     BD.RCH(R3),RCVQIO+Q.IOPL+4  ; Set receive char address
        MOV     #RC.LEN,RCVQIO+Q.IOPL+6     ;              and length
        DIR$    #RCVQIO                 ; Hang another receive
        BCC     100$                    ; If CC, receive is queued
```

```
90$:    DSAR$S                          ; Disable AST recognition
        SETF$S  #DONE                   ; Wake up mainline for cleanup

100$:   RESRG   <R5,R4,R3>
        ASTX$S

        .SBTTL  .       ABOAST  - Abort AST
ABOAST:
        ADD     (SP),SP
        DSAR$S                                  ; Disable AST recognition
        SETF$S  #DONE                           ; Done
        ASTX$S

        .SBTTL  Utility subroutines
        .SBTTL  .       ASNLNS - Assign channel, command, error LUNs

ASNLNS:
;
; Assign LUN to channel
;
        ALUN$S  #CHNLUN,#"NX,0
        BCC     10$
        JSR     R0,DIRERR
        .WORD   ASNFM1
        BR      40$
;
; Assign LUN to command terminal
;
10$:
        ALUN$S  #CMDLUN,#"TI
        BCC     20$
        JSR     R0,DIRERR
        .WORD   ASNFM2
        BR      40$
;
; Assign LUN to output terminal
;
20$:    ALUN$S  #TILUN,#"TI
        BCC     30$
        JSR     R0,DIRERR
        .WORD   ASNFM3
        BR      40$
30$:    TST     (PC)+
40$:    SEC
        RETURN

        .SBTTL  .       OPNLOG- Open a log file

        .ENABL  LSB
OPNLOG:
        BIT     #OP.LOG,OPTFLG          ; /LOG requested?
        BEQ     20$                     ; If EQ, no
        OPEN$A  #LOGFIL                 ; Append to file if existing
        BCC     10$                     ; If CC, file is open
        OPEN$W  #LOGFIL                 ; Else, create a new file
```

(continued on next page)

```
        BCC     10$                     ; If CC, file is open
        BIC     #OP.LOG,OPTFLG          ; Indicate no log file in use
        BR      20$                     ;   and return
10$:    GTIM$S  #TIMBUF                 ; Get current time
        FORMAT  #TIMBUF,#TIMFMT         ; Format time into ASCII
        PUT$    #LOGFIL,#FMTBUF,FMTL    ; Store timestamp in log file
        SUB     #CMDBUF,R4              ; Compute length of command
        PUT$    #LOGFIL,#CMDBUF,R4      ;   and log the command line
20$:    RETURN
        .DSABL  LSB

        .SBTTL  .       OPNCHN - Open channel

OPNCHN:
        MOV     .DEVNM+2,OPNQIO+Q.IOPL  ; Set "xxx-n" address
        MOV     .DEVNM,OPNQIO+Q.IOPL+2  ; Set "xxx-n" length
        MOV     #TMO,TIMOUT             ; Set timeout in seconds
        BIT     #OP.RSP,OPTFLG          ; Are we passively responding?
        BEQ     5$                      ; If EQ, no
        CLR     TIMOUT                  ; No timeout on receives
5$:     MOV     TIMOUT,OPNQIO+Q.IOPL+4  ; Set the time-out for open
        MOV     #OPNCHL,OPNQIO+Q.IOPL+10; Assume Class I service
        BIT     #OP.USR,OPTFLG          ; Opening for Class I service?
        BEQ     7$                      ; If EQ, yes
        MOV     #OPNCL0,OPNQIO+Q.IOPL+10; Else get set for user service
7$:     DIR$    #OPNQIO                 ; Open the line
        BCC     10$                     ; If, CC, OPEN queued
        JSR     R0,DIRERR
        .WORD   OPNFMT
        BR      20$

10$:    TSTB    CHNSB                   ; Did OPEN succeed?
        CLC                             ; Assume yes
        BPL     30$                     ; If PL, yes
        JSR     R0,IOERR
        .WORD   OPNFMT
20$:    ERROR   #OPNCHB,#OPNCHF
        SEC                             ; Indicate failure

30$:    RETURN

        .SBTTL          SETCHN - Set channel characteristics

SETCHN: MOVB    .ISAP,SETISP            ; Store SAP number
        MOV     #SETBF1,SETQIO+Q.IOPL   ; Get char buffer address
        MOV     #SETLN1,SETQIO+Q.IOPL+2 ; Get char buffer length
        DIR$    #SETQIO                 ; Enable the ISAP
        BCS     20$                     ; If CC, request got queued
        TSTB    CHNSB                   ; Did we enable an ISAP?
        BMI     30$                     ; If MI, no

        BIT     #OP.GSP,OPTFLG          ; Enabling a group SAP?
        BEQ     10$                     ; If EQ, no
        MOVB    .GSAP,SETGSP            ; Store group SAP number
        MOV     #SETBF2,SETQIO+Q.IOPL   ; Get char buffer address
```

```
        MOV     #SETLN2,SETQIO+Q.IOPL+2 ; Get char buffer length
        DIR$    #SETQIO                 ; Enable the GSAP
        BCS     20$                     ; If CC, request got queued
        TSTB    CHNSB                   ; Did we enable a GSAP?
        BMI     30$                     ; If MI, no

10$:    BIT     #OP.SNP,OPTFLG          ; Enabling a SNAP protocol ID?
        CLC                             ; Assume not
        BEQ     50$                     ; If EQ, all done
        MOV     #.SNAP,R0               ; Copy protocol ident to buffer
        MOV     #SETSNA,R1              ;
        MOV     #5,R2                   ;
15$:    MOVB    (R0)+,(R1)+             ;
        SOB     R2,15$                  ;
        MOV     #SETBF3,SETQIO+Q.IOPL   ; Get char buffer address
        MOV     #SETLN3,SETQIO+Q.IOPL+2 ; Get char buffer length
        DIR$    #SETQIO                 ; Enable the SNAP protocol ID
        BCS     20$                     ; If CC, request got queued
        TSTB    CHNSB                   ; Did we enable a SNAP?
        CLC.                            ; Assume success
        BPL     50$                     ; If PL, yes - all done
        BMI     30$                     ; If MI, no

20$:    JSR     R0,DIRERR
        .WORD   SETFMT
        BR      40$
30$:    JSR     R0,IOERR
        .WORD   SETFMT
40$:    SEC                             ; Indicate failure
50$:    RETURN

        .SBTTL  .       LOGRCV - Log received data
;+
; **-LOGRCV - Log received data
;
; Inputs:
;       R3 is buffer descriptor block address
;               BD.BUF(R3) contains the buffer address
;               BD.STS+2(R3) contains the number of bytes received
;       R4 is received characteristics block address
;               RC.DAD(R4) contains the received destination address
;               RC.SAD(R4)                       source
;               RC.SPM(R4)                       DSAP/SSAP
;               RC.CTM(R4)                       CTL
;
; Registers modified:
;       R4,R5
;-

LOGRCV: TST     BD.BUF+2(R3)                    ; Any data to log?
        BNE     10$                             ; If NE, yes
        JMP     100$                            ; Else, get out
10$:    SAVRG   <R0,R1,R2>
        LOG     #RCVMS1,#RCVLN1                 ; Log DST Ethernet address
        ADD     #RC.DAD,R4                      ;
```

(continued on next page)

```
        MOV     #6,R5                           ;
        CALL    LOGDAT                          ;
        SUB     #RC.DAD,R4                      ;

        LOG     #RCVMS2,#RCVLN2                 ; Log SRC Ethernet address
        ADD     #RC.SAD,R4                      ;
        CALL    LOGDAT                          ;
        SUB     #RC.SAD,R4                      ;

        BIT     #OP.SNP,OPTFLG                  ; Logging DSAP/SSAP/CTL?
        BNE     20$                             ; If NE, no - log a SNAP id

        LOG     #RCVMS3,#RCVLN3                 ; Log DSAP/SSAP/CTL bytes
        MOV     #FMTDAT,R2                      ;
        MOVB    RC.SPM(R4),(R2)+                ;
        BICB    #1,RC.SPM+1(R4)                 ;
        MOVB    RC.SPM+1(R4),(R2)+              ;
        MOVB    RC.CTM(R4),(R2)+                ;
        MOV     #FMTDAT,R4                      ;
        MOV     #3,R5                           ;
        CALL    LOGDAT                          ;
        BR      30$                             ;

20$:    LOG     #RCVMS4,#RCVLN4                 ; Log SNAP protocol ID
        MOV     #FMTDAT,R2                      ;
        ADD     #RC.SNM,R4                      ;
        MOV     #5,R5                           ;
        CALL    LOGDAT                          ;

30$:    MOV     BD.STS+2(R3),R5                 ; Received data, n. bytes
        MOV     R5,FMTDAT                       ;
        FORMAT  #FMTDAT,#RCVMS5                 ;
        LOG     #FMTBUF,FMTL                    ;
        MOV     BD.BUF(R3),R4                   ;
        CALL    LOGDAT                          ;

        RESRG   <R2,R1,R0>
100$:   RETURN

        .SBTTL  .       DIRERR - Report a directive error
        .SBTTL  .       IOERR  - Report an I/O error
        .SBTTL  .       IOER2  - Report an I/O error (alternate entry)
        .ENABL  LSB
DIRERR:
        MOV     (R0)+,FMTDAT
        SAVRG   <R0,R1,R2>
        MOV     $DSW,FMTDAT+2
        ERROR   #FMTDAT,#ERRFM1
        BR      10$

IOERR:
        MOV     (R0)+,FMTDAT
        SAVRG   <R0,R1,R2>
        MOV     CHNSB,FMTDAT+2
        MOV     CHNSB+2,FMTDAT+4
```

```
        ERROR   #FMTDAT,#ERRFM2
        BR      10$

IOER2:
        MOV     (R0)+,FMTDAT
        SAVRG   <R0,R1,R2>
        MOVB    RCVSB,R1
        MOV     R1,FMTDAT+2
        MOV     RCVSB+2,FMTDAT+4
        ERROR   #FMTDAT,#ERRFM2

10$:    MOV     #EX$ERR,EXSTAT
        RESRG   <R2,R1,R0>
        RTS     R0
        .DSABL  LSB

        .SBTTL  .       LOGDAT - Log data in hex

HEX     = 16.
NOSUP   = 1*1000
BLKFIL  = 1*2000
FLDWID  = 2*4000
MASK    = HEX+FLDWID+NOSUP
NFLDS   = 16.

LOGDAT: CALL    $SAVAL

10$:    MOV     #FMTBUF,R0
        MOV     #NFLDS,R3
        CMP     R3,R5
        BLE     20$
        MOV     R5,R3
        BEQ     50$

20$:    MOV     R3,-(SP)
        MOVB    #40,(R0)+
        MOVB    #40,(R0)+
        MOVB    #40,(R0)+
        MOVB    #40,(R0)+
30$:    DEC     R3
        BLT     40$

        CLR     R1
        BISB    (R4)+,R1
        MOV     #MASK,R2
        CALL    $CBTA
        MOVB    #40,(R0)+

        BR      30$
40$:    SUB     #FMTBUF,R0
        MOV     R0,R1
        LOG     #FMTBUF,R1
        SUB     (SP)+,R5
        BGT     10$
```

(continued on next page)

```
50$:    RETURN

        .SBTTL  Parser data base

        $RONLY  = 1                                     ; Make tables read-only
        ISTAT$  STATBL,KEYTBL

        .SBTTL  .       Main states

        STATE$  START
         TRAN$  $LAMDA

        .SBTTL  .       Options

        STATE$  OPT
         TRAN$  $EOS,$EXIT
         TRAN$  <';>,$EXIT
         TRAN$  '/
        STATE$
         TRAN$  !DEVOPT,OPT,,OP.DEV,OPTFLG
         TRAN$  !DSPOPT,OPT,,OP.DSP,OPTFLG
         TRAN$  !GSPOPT,OPT,,OP.GSP,OPTFLG
         TRAN$  !HDWOPT,OPT,,OP.HDW,OPTFLG
         TRAN$  !ISPOPT,OPT,,OP.ISP,OPTFLG
         TRAN$  !LOGOPT,OPT,,OP.LOG,OPTFLG
         TRAN$  !MSGOPT,OPT,,OP.MSG,OPTFLG
         TRAN$  !NODOPT,OPT,,OP.NOD,OPTFLG
         TRAN$  !PHYOPT,OPT,,OP.PHY,OPTFLG
         TRAN$  !RSPOPT,OPT,,OP.RSP,OPTFLG
         TRAN$  !SIZOPT,OPT,,OP.SIZ,OPTFLG
         TRAN$  !SNPOPT,OPT,,OP.SNP,OPTFLG
         TRAN$  !USROPT,OPT,,OP.USR,OPTFLG

        .SBTTL  .       .  DEVOPT - /DEV[ICE]=ddd-n

        STATE$  DEVOPT
         TRAN$  "DEVICE"
        STATE$
         TRAN$  EQUALS
        STATE$
         TRAN$  !DEVICE,$EXIT,STDEV

        .SBTTL  .       .  DSPOPT - /DSAP={n,NULL,SNAP}

        STATE$  DSPOPT
         TRAN$  "DSAP"
        STATE$
         TRAN$  EQUALS
        STATE$
         TRAN$  !DSPID,$EXIT

        .SBTTL  .       .  GSPOPT - /GSA[P]=n

        STATE$  GSPOPT
         TRAN$  "GSAP"
```

```
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !GSPID,$EXIT

.SBTTL  .       .  HDWOPT - /HAR[DWARE]=nn-nn-nn-nn-nn-nn

STATE$  HDWOPT
 TRAN$  "HARDWARE"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !HXADR,$EXIT,STHADD

.SBTTL  .       .  ISPOPT - /ISAP=n

STATE$  ISPOPT
 TRAN$  "ISAP",ISPOP2
 TRAN$  "SSAP"
STATE$  ISPOP2
 TRAN$  EQUALS
STATE$
 TRAN$  !ISPID,$EXIT

.SBTTL  .       .  LOGOPT - /LOG

STATE$  LOGOPT
 TRAN$  "LOG",$EXIT

.SBTTL  .       .  MSGOPT - /MES[SAGE]={XID,TEST,UI}

STATE$  MSGOPT
 TRAN$  "MESSAGE"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !MSGTYP,$EXIT

.SBTTL  .       .  NODOPT - /NOD[E]=aa.nn

STATE$  NODOPT
 TRAN$  "NODE"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !NODID,$EXIT,

.SBTTL  .       .  PHYOPT - /PHY[SICAL]=nn-nn-nn-nn-nn-nn

STATE$  PHYOPT
 TRAN$  "PHYSICAL"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !HXADR,$EXIT,STPADD
```

(continued on next page)

```
.SBTTL  .       .  RSPOPT - /RES[PONSE]

STATE$  RSPOPT
 TRAN$  "RESPONDER",$EXIT

.SBTTL  .       .  SIZOPT - /SIZ[E]=n

STATE$  SIZOPT
 TRAN$  "SIZE"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !SIZE,$EXIT

.SBTTL  .       .  SNPOPT - /SNA[P]=nn-nn-nn

STATE$  SNPOPT
 TRAN$  "SNAP"
STATE$
 TRAN$  EQUALS
STATE$
 TRAN$  !SNAP,$EXIT

.SBTTL  .       .  USROPT - /USE[R]

STATE$  USROPT
 TRAN$  "USER",$EXIT

.SBTTL  .       Utility substates
.SBTTL  .       .  DEVICE - device string

STATE$  DEVICE
 TRAN$  $RAD50
STATE$
 TRAN$  <'->
STATE$
 TRAN$  $DNUMB,$EXIT

.SBTTL  .       .  DSPID  - destination SAP number

STATE$  DSPID
 TRAN$  "SNAP",$EXIT,STSNP
 TRAN$  "NULL",$EXIT,STNSP
 TRAN$  $DNUMB,$EXIT,STDSP

.SBTTL  .       .  GSPID  - group SAP number

STATE$  GSPID
 TRAN$  $DNUMB,$EXIT,STGSP

.SBTTL  .       .  HXADR  - hex address

STATE$  HXADR
 TRAN$  !HXBYT,,STHAD1
```

```
STATE$
 TRAN$   '-
STATE$
 TRAN$   !HXBYT,,STHAD2
STATE$
 TRAN$   '-
STATE$
 TRAN$   !HXBYT,,STHAD3
STATE$
 TRAN$   '-
STATE$
 TRAN$   !HXBYT,,STHAD4
STATE$
 TRAN$   '-
STATE$
 TRAN$   !HXBYT,,STHAD5
STATE$
 TRAN$   '-
STATE$
 TRAN$   !HXBYT,$EXIT,STHAD6

.SBTTL   .       .  HXBYT  - hex byte

STATE$   HXBYT
 TRAN$   !HXDIG,,STHXD1
STATE$
 TRAN$   !HXDIG,$EXIT,STHXD2

.SBTTL   .       .  HXDIG  - hex digit

STATE$   HXDIG
 TRAN$   $DIGIT,$EXIT,STHXN
 TRAN$   $ALPHA,$EXIT,STHXA

.SBTTL   .       .  ISPID  - Individual SAP number

STATE$   ISPID
 TRAN$   $DNUMB,$EXIT,STISP

.SBTTL   .       .  MSGTYP - message type

STATE$   MSGTYP
 TRAN$   "TEST",$EXIT,STTST,OP.TST,MSGFLG
 TRAN$   "UI",$EXIT,STUIF,OP.UIF,MSGFLG
 TRAN$   "XID",$EXIT,STXID,OP.XID,MSGFLG

.SBTTL   .       .  NODID  - node id

STATE$   NODID
 TRAN$   $DNUMB,,STNDA
STATE$
 TRAN$   <'.>
STATE$
 TRAN$   $DNUMB,$EXIT,STNDN
```

(continued on next page)

```
        .SBTTL  .       .  SIZE   - data block size

        STATE$  SIZE
         TRAN$  $DNUMB,$EXIT,STSIZ

        .SBTTL  .       .  SNAP   - SNAP protocol identifier

        STATE$  SNAP
         TRAN$  !HXBYT,,STSNP1
        STATE$
         TRAN$  '-
        STATE$
         TRAN$  !HXBYT,,STSNP2
        STATE$
         TRAN$  '-
        STATE$
         TRAN$  !HXBYT,,STSNP3
        STATE$
         TRAN$  '-
        STATE$
         TRAN$  !HXBYT,,STSNP4
        STATE$
         TRAN$  '-
        STATE$
         TRAN$  !HXBYT,$EXIT,STSNP5

        STATE$

        .SBTTL  Parser action routines
        .PSECT  $CODE

        .SBTTL  .       STNDA  - Set node area
        .SBTTL  .       STNDN  - Set node number

        .ENABL  LSB
STNDA:  MOV     .PNUMB,R0               ; Get area number
        TST     .PNUMH                  ; Overflow into high word?
        BNE     10$                     ; If NE, yes - error
        CMP     R0,#MXAREA              ; Area in range?
        BHI     10$                     ; If HI, no - error
        MOV     R0,.NODID               ; Store area number
        BR      20$                     ;   and return
STNDN:  MOV     .PNUMB,R0               ; Get node number
        TST     .PNUMH                  ; Overflow into high word?
        BNE     10$                     ; If NE, yes - error
        CMP     R0,#MXNODE              ; Number in range?
        BHI     10$                     ; If HI, no - error
        MOV     R0,.NODID+2             ; Store node number

        BIT     #OP.PHY,OPTFLG          ; Already specified physical address?
        BNE     20$                     ; If NE, yes - use it

        MOVB    #252, .PHADR+0
        MOVB    #0,   .PHADR+1
        MOVB    #4,   .PHADR+2
```

```
        MOVB    #0,   .PHADR+3
; Set up area and number in .PHADR+4, +5 ...
        MOV     .NODID,R0               ; Get area in R0
        MOV     .NODID+2,R1             ; Get node number in R1
        SWAB    R0                      ; Get area in high byte
        ASL     R0                      ; Move area into <15:10>
        ASL     R0                      ;
        BIS     R0,R1                   ; Form node address word
        MOVB    R1,.PHADR+4             ;   and store in .PHADR+4,+5
        SWAB    R1                      ;
        MOVB    R1,.PHADR+5             ;
        BR      20$                     ; Join common code for return
10$:    ADD     #2,(SP)                 ; REJECT TRANSITION
20$:    RETURN
        .DSABL  LSB

        .SBTTL  .       STDEV  - Set device string
        .SBTTL  .       STSNP  - Set 802.3 SNAP SAP as DSAP
        .SBTTL  .       STNSP  - Set 802.3 NULL SAP as DSAP
        .SBTTL  .       STDSP  - Set 802.3 user-specified SAP as DSAP
        .SBTTL  .       STISP  - Set 802.3 user-specified SAP as ISAP
        .SBTTL  .       STSIZ  - Set data block size

STDEV:  MOV     .PSTCN,.DEVNM
        MOV     .PSTPT,.DEVNM+2
        RETURN

        .ENABL  LSB
STSNP:  MOVB    #^B10101010,.DSAP       ; Store the SNAP SAP
        BR      20$
STNSP:  CLRB    .DSAP                   ; Store the NULL SAP
        BR      20$
STDSP:  CALL    30$                     ; Is SAP in range?
        BCS     10$                     ; If CS, no
        MOVB    .PNUMB,.DSAP            ; Store destination SAP
        BR      20$
STGSP:  CALL    30$                     ; Is SAP in range?
        BCS     10$                     ; If CS, no
        BIT     #1,.PNUMB               ; Valid group?
        BEQ     10$                     ; If EQ, no - it's individual
        MOVB    .PNUMB,.GSAP            ; Store Group SAP
        BIS     #OP.USR,OPTFLG          ; User-supplied service is needed
        BR      20$
STISP:  CALL    30$                     ; Is SAP in range?
        BCS     10$                     ; If CS, no
        BIT     #1,.PNUMB               ; Is this an Individual SAP?
        BNE     10$                     ; If NE, no - it's a group
        MOVB    .PNUMB,.ISAP            ; Else, store individual SAP
        BR      20$
STSIZ:  CMP     .PNUMB,#MAXFRM          ; Data block size too large?
        BHI     10$                     ; If HI, yes
        TST     .PNUMH                  ; Is it?
        BNE     10$                     ; If NE, yes
        MOV     .PNUMB,.SIZE            ; Store data block size
        BIT     #OP.TST!OP.UIF,MSGFLG   ; Already parse message type?
```

(continued on next page)

```
        BEQ     20$                     ; If EQ, no - .SIZE will get used
        MOV     .SIZE,.MSLEN            ; Else, stuff size into message len
        BR      20$
10$:    ADD     #2,(SP)                 ; REJECT TRANSITION
20$:    RETURN
;
; Check if SAP is in range
;
30$:    CMP     .PNUMB,#377             ; Is SAP in range?
        BHI     40$                     ; If HI, no
        TST     .PNUMH                  ; Is it?
        BNE     40$                     ; If NE, no
        TST     (PC)+                   ; SUCCESS (C=0)
40$:    SEC
        RETURN
        .DSABL  LSB

        .SBTTL  .       STTST  - Set TEST message type
        .SBTTL  .       STUIF  - Set UI   message type
        .SBTTL  .       STXIF  - Set XID  message type

STTST:  MOVB    #$CSTSF,.CTL
        MOV     #TSTMSG,.MSADR
        MOV     .SIZE,.MSLEN
        RETURN

STUIF:  MOVB    #$CSUIF,.CTL
        MOV     #UIFMSG,.MSADR
        MOV     .SIZE,.MSLEN
        RETURN

STXID:  MOVB    #$CSXIF,.CTL
        MOV     #XIDMSG,.MSADR
        MOV     #XIDLEN,.MSLEN
        RETURN

        .SBTTL  .       STSNPn - Set SNAP protocol ID (byte #n)

STSNP1: MOVB    .HXBYT,.SNAP
STSNP2: MOVB    .HXBYT,.SNAP+1
STSNP3: MOVB    .HXBYT,.SNAP+2
STSNP4: MOVB    .HXBYT,.SNAP+3
STSNP5: MOVB    .HXBYT,.SNAP+4
        RETURN

        .SBTTL  .       STPADD - Set physical address
        .SBTTL  .       STHADD - Set hardware address

        .ENABL  LSB
STPADD: BIT     #<OP.NOD!OP.PHY>,OPTFLG ; Already specified remote address?
        BNE     10$                     ; If NE, yes - all done
        MOV     .HXADR,.PHADR
        MOV     .HXADR+2,.PHADR+2
        MOV     .HXADR+4,.PHADR+4
        RETURN
```

```
STHADD: BIT     #<OP.PHY!OP.NOD>,OPTFLG ; Already specified remote address?
        BNE     10$                     ;
        MOV     .HXADR,.HWADR
        MOV     .HXADR+2,.HWADR+2
        MOV     .HXADR+4,.HWADR+4
10$:    RETURN
        .DSABL  LSB

        .SBTTL  .       STHADn - Set hex address (byte #n)

STHAD1: MOVB    .HXBYT,.HXADR
STHAD2: MOVB    .HXBYT,.HXADR+1
STHAD3: MOVB    .HXBYT,.HXADR+2
STHAD4: MOVB    .HXBYT,.HXADR+3
STHAD5: MOVB    .HXBYT,.HXADR+4
STHAD6: MOVB    .HXBYT,.HXADR+5
        RETURN

        .SBTTL  .       STHXN  - Convert a digit to hex
        .SBTTL  .       STHXA  - Convert an alpha to hex

STHXN:  MOVB    .PCHAR,R0               ; Get digit character
        SUB     #'0,R0                  ; Convert to digit value
        MOVB    R0,.HXDIG               ;   and store
        RETURN

STHXA:  MOVB    .PCHAR,R0               ; Get alpha character
        CMPB    R0,#'A                  ; Is it a hex digit?
        BLO     10$                     ; If LO, no
        CMPB    R0,#'F                  ; Is it?
        BLOS    20$                     ; If LOS, yes
10$:    ADD     #2,(SP)                 ; Reject transition
        BR      30$                     ;   and return
20$:    SUB     #<'A-10.>,R0            ; Convert to value
        MOVB    R0,.HXDIG
30$:    RETURN

        .SBTTL  .       STHXD1 - Set 1st hex digit
        .SBTTL  .       STHXD2 - Set 2nd hex digit

        .ENABL  LSB
STHXD1: MOVB    .HXDIG,R0
        ASL     R0
        ASL     R0
        ASL     R0
        ASL     R0
        MOVB    R0,.HXBYT
        BR      10$
STHXD2: MOVB    .HXDIG,R0
        BICB    #^C17,R0
        BISB    R0,.HXBYT
10$:    RETURN
        .DSABL  LSB

        .SBTTL  .       PRSDMP - Dump parse data on syntax error
```

(continued on next page)

```
;DMP: CALL    $SAVAL
     MOV     #FMTDAT,R5             ; Point at binary buffer
     MOV     R3,(R5)+               ; Store unparsed string length
     MOV     R4,(R5)+               ;                        address
     MOV     OPTFLG,(R5)+           ; Store option flags
     MOV     MSGFLG,(R5)+           ; Store message flags

     MOV     .DEVNM,(R5)+           ; Store device name length
     MOV     .DEVNM+2,(R5)+         ;                    address
     MOV     .NDADR,(R5)+           ; Store destination Ethernet addr
     MOV     .NDADR+2,(R5)+         ;
     MOV     .NDADR+4,(R5)+         ;

     MOVB    .ISAP,(R5)+            ; Store individual SAP
     CLRB    (R5)+                  ;   as a word
     MOVB    .GSAP,(R5)+            ; Store group SAP
     CLRB    (R5)+                  ;   as a word
     MOVB    .DSAP,(R5)+            ; Store destination SAP
     CLRB    (R5)+                  ;   as a word
     MOV     .CTL,(R5)+             ; Store CTL field

     MOV     #.SNAP,(R5)+           ; Store SNAP protocol address

     FORMAT  #FMTDAT,#PRSERR        ; Format data into ASCII
     LOG     #FMTBUF,FMTL           ; Log the text
     RETURN

     .END    TST802
```

4.4.7.2 Ethernet Example

This program uses Ethernet frame format. You can use the program to remotely trigger a QNA controller.

```
        .TITLE  TRGQNA - Trigger QNA
        .IDENT  /X1.01/
        .NLIST  BEX
;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
        .SBTTL  Program Description
;
;       This program enables remote triggering of a properly configured
;       PDP-11 Q-bus system with a DEQNA that is running RSX-11S with DECnet.
;       This enables an operator to load a new system image into a running
;       system without manually rebooting the system.
;
;       If the system hangs or crashes the system will request a reboot
;       from the network via the DEQNA.
;
;       Notes and Cautions:
;
;       - The processor board must be either a KDF11-BE/BF with the KDF11-B2
;         bootstrap ROM update kit installed or a KDJ11-B.
;
;       - The bootstrap switches (1-8) on the processor board must be set so
;         that the boot request will be directed to the DEQNA.
;
;            Switch
;       8 7 6 5 4 3 2 1
;       ---------------
;       x x 0 1 0 1 0 0 . . . . DEQNA unit 0
;       x x 0 1 0 1 0 1 . . . . DEQNA unit 1
;       0 x x x x x x x . . . . Console terminal is not an ANSI mode scope
;       1 x x x x x x x . . . . Console terminal is an ANSI mode scope
;       x 0 x x x x x x . . . . Loop self test but no memory diagnostics
;       x 1 x x x x x x . . . . Loop self test and memory diagnostics
;
;
;       Instructions for building the task:
;
```

(continued on next page)

```
        .WORD   0                       ;C.CHRL = timer value (Off)
STMLN   = .-STMBUF                      ;Length for set characteristics
;
; DPB for write to terminal
;
OUTIO:  QIOW$   IO.WVB,TILUN,2,,,,<0,0,40>

RCVBUF: .BLKB   100                     ;Receive buffer
RCVLN   = .-RCVBUF                      ;Length of receive buffer

IOSB:   .BLKW   2                       ;I/O status block for QIOs

        .SBTTL  Text Strings

;
; Information / Error Messages
;
ASNERR: .ASCIZ  /TRG - Unable to assign LUN/
OPNERR: .ASCIZ  /TRG - Unable to open line/
DSTERR: .ASCIZ  /TRG - Unable to set protocol type or sanity timer on/
RCVERR: .ASCIZ  /TRG - Unable to request receive data/
STMERR: .ASCIZ  /TRG - Unable to set sanity timer off/
REBOOT: .ASCIZ  /TRG - *** Remote trigger received -- system re-booting ***/
;
        .EVEN
LINE:   .ASCII  /QNA-0/                 ;Line definition for Set Characteristi
LINL    =       .-LINE                  ;Line definition length
        .EVEN

        .SBTTL  Main Line Code

START::
        ALUN$S  #TILUN,#"CO,#0          ;Assign LUN to terminal
        ALUN$S  #DLXLUN,#"NX,#0         ;Assign a LUN to DLX
        BCC     10$                     ;If CC, ok
        MOV     #ASNERR,R0              ;Print assign LUN error msg
        CALL    PRINT                   ;...
        BR      999$                    ; and exit
;
; Open the line
;
10$:
        MOV     #IOSB,R3                ;Get address of I/O status block
        MOV     #LINE,R0                ;Get address of line to be opened
        MOV     #LINL,R1                ;Get length of line descriptor
        QIOW$S  #IO.XOP,#DLXLUN,#1,,R3,,<R0,R1,#400>
        BCS     20$                     ;If CS, directive error
        MOVB    (R3),R1                 ;Get "OPEN" QIO status
        BPL     30$                     ;Plus, is ok
20$:
        MOV     #OPNERR,R0              ;Print open error msg
        CALL    PRINT                   ;...
        BR      99$                     ; and exit
;
; Enable console carrier protocol type and set sanity timer on
```

```
;
; Note that the QNA driver will refresh the timer as long as it is running
;
30$:
        MOV     #IOSB,R3                ;Get I/O status block address
        MOV     #DSTBUF,R4              ;Get address of characteristics buffer
        MOV     #DSTLN,R5               ;Get length of characteristics buffer
        CALL    SETCHR                  ;Set characteristics
        BCC     40$                     ;If CC, success
        MOV     #DSTERR,R0              ;Print set characteristics error msg
        CALL    PRINT                   ;...
        BR      99$                     ;And exit
;
; Hang a receive to look for trigger message
;
40$:
        QIOW$S  #IO.XRC,#DLXLUN,#1,,#IOSB,,<#RCVBUF,#RCVLN>
        BCS     50$                     ;If CC, directive success
        TSTB    IOSB                    ;Was the receive successful?
        BPL     60$
50$:
        MOV     #RCVERR,R0              ;Print receive error msg
        CALL    PRINT
        BR      99$                     ;And exit
;
; Make sure message received is trigger message
;
60$:
        MOV     #RCVBUF,R0              ;Get the message address
        MOV     IOSB+2,R1               ;Get the message length
        CMP     R1,#5                   ;Is the message minimum length?
        BLT     40$                     ;If LT, no - try for another message
        CMPB    #6,(R0)+                ;Is this a boot message?
        BNE     40$                     ;If NE, no - try for another message
        MOV     #REBOOT,R0              ;Print re-boot message
        CALL    PRINT                   ;...
;
; Set the sanity timer off
;
        MOV     #IOSB,R3                ;Get I/O status block address
        MOV     #STMBUF,R4              ;Get address of characteristic s buffer
        MOV     #STMLN,R5               ;Get length of characteristics  buffer
        CALL    SETCHR                  ;Set characteristics
        BCC     70$                     ;If CC, success
        MOV     #STMERR,R0              ;Print set characteristics er ror msg
        CALL    PRINT                   ;...
        BR      99$                     ;And exit

;
; Switch to system state and jump to system boot
;
70$:
        CALL    $SWSTK,99$              ;Switch to system state
        JMP     @#173000                ;Activate system boot
;
```

(continued on next page)

```
; Error exit
;
99$:
        QIOW$S  #IO.XCL,#DLXLUN,#1      ;Close the open line
999$:
        EXIT$S                          ;Exit the task

;
; Set characteristics routine
;
SETCHR:
        QIOW$S  #IO.XSC,#DLXLUN,#1,,R3,,<R4,R5> ;Issue set characteristics
        BCS     10$                     ;If CS, directive error
        TSTB    (R3)                    ;Any problem with the QIO?
        BMI     10$                     ;IF MI, yes
        CMP     #CS.SUC,6(R4)           ;Any problem with characteristics?
        BNE     10$                     ;If NE, yes
        TST     (PC)+                   ;Indicate success
10$:    SEC                             ;Indicate error
        RETURN                          ;Return to caller
;
; Print message routine
;
PRINT:
        MOV     R0,OUTIO+Q.IOPL         ;Save message address in DPB
5$:
        TSTB    (R0)+                   ;Search for end of message
        BNE     5$                      ; (terminated by null)
        DEC     R0                      ;Back up to null
        SUB     OUTIO+Q.IOPL,R0         ;Compute length of message
        MOV     R0,OUTIO+Q.IOPL+2       ; and save in DPB
        DIR$    #OUTIO                  ;Print error message
        MOV     #40,OUTIO+Q.IOPL+4      ;Set up carriage control
        RETURN                          ;Return to caller

        .END    START
        BEQ     20$                     ; If EQ, no - .SIZE will get used
        MOV     .SIZE,.MSLEN            ; Else, stuff size into message len
        BR      20$
10$:    ADD     #2,(SP)                 ; REJECT TRANSITION
20$:    RETURN
;
; Check if SAP is in range
;
30$:    CMP     .PNUMB,#377             ; Is SAP in range?
        BHI     40$                     ; If HI, no
        TST     .PNUMH                  ; Is it?
        BNE     40$                     ; If NE, no
        TST     (PC)+                   ; SUCCESS (C=0)
40$:    SEC
        RETURN
        .DSABL  LSB

        .SBTTL  .       STTST  - Set TEST message type
        .SBTTL  .       STUIF  - Set UI   message type
        .SBTTL  .       STXIF  - Set XID  message type
```

```
STTST:  MOVB    #$CSTSF,.CTL
        MOV     #TSTMSG,.MSADR
        MOV     .SIZE,.MSLEN
        RETURN

STUIF:  MOVB    #$CSUIF,.CTL
        MOV     #UIFMSG,.MSADR
        MOV     .SIZE,.MSLEN
        RETURN

STXID:  MOVB    #$CSXIF,.CTL
        MOV     #XIDMSG,.MSADR
        MOV     #XIDLEN,.MSLEN
        RETURN

        .SBTTL  .       STSNPn - Set SNAP protocol ID (byte #n)

STSNP1: MOVB    .HXBYT,.SNAP
STSNP2: MOVB    .HXBYT,.SNAP+1
STSNP3: MOVB    .HXBYT,.SNAP+2
STSNP4: MOVB    .HXBYT,.SNAP+3
STSNP5: MOVB    .HXBYT,.SNAP+4
        RETURN

        .SBTTL  .       STPADD - Set physical address
        .SBTTL  .       STHADD - Set hardware address

        .ENABL  LSB
STPADD: BIT     #<OP.NOD!OP.PHY>,OPTFLG ; Already specified remote address?
        BNE     10$                     ; If NE, yes - all done
        MOV     .HXADR,.PHADR
        MOV     .HXADR+2,.PHADR+2
        MOV     .HXADR+4,.PHADR+4
        RETURN

STHADD: BIT     #<OP.PHY!OP.NOD>,OPTFLG ; Already specified remote address?
        BNE     10$                     ;
        MOV     .HXADR,.HWADR
        MOV     .HXADR+2,.HWADR+2
        MOV     .HXADR+4,.HWADR+4
10$:    RETURN
        .DSABL  LSB

        .SBTTL  .       STHADn - Set hex address (byte #n)

STHAD1: MOVB    .HXBYT,.HXADR
STHAD2: MOVB    .HXBYT,.HXADR+1
STHAD3: MOVB    .HXBYT,.HXADR+2
STHAD4: MOVB    .HXBYT,.HXADR+3
STHAD5: MOVB    .HXBYT,.HXADR+4
STHAD6: MOVB    .HXBYT,.HXADR+5
        RETURN

        .SBTTL  .       STHXN  - Convert a digit to hex
        .SBTTL  .       STHXA  - Convert an alpha to hex
```

(continued on next page)

```
STHXN:  MOVB    .PCHAR,R0               ; Get digit character
        SUB     #'0,R0                  ; Convert to digit value
        MOVB    R0,.HXDIG               ;   and store
        RETURN

STHXA:  MOVB    .PCHAR,R0               ; Get alpha character
        CMPB    R0,#'A                  ; Is it a hex digit?
        BLO     10$                     ; If LO, no
        CMPB    R0,#'F                  ; Is it?
        BLOS    20$                     ; If LOS, yes
10$:    ADD     #2,(SP)                 ; Reject transition
        BR      30$                     ;   and return
20$:    SUB     #<'A-10.>,R0            ; Convert to value
        MOVB    R0,.HXDIG
30$:    RETURN

        .SBTTL  .       STHXD1 - Set 1st hex digit
        .SBTTL  .       STHXD2 - Set 2nd hex digit

        .ENABL  LSB
STHXD1: MOVB    .HXDIG,R0
        ASL     R0
        ASL     R0
        ASL     R0
        ASL     R0
        MOVB    R0,.HXBYT
        BR      10$
STHXD2: MOVB    .HXDIG,R0
        BICB    #^C17,R0
        BISB    R0,.HXBYT
10$:    RETURN
        .DSABL  LSB

        .SBTTL  .       PRSDMP - Dump parse data on syntax error

PRSDMP: CALL    $SAVAL
        MOV     #FMTDAT,R5              ; Point at binary buffer
        MOV     R3,(R5)+                ; Store unparsed string length
        MOV     R4,(R5)+                ;                        address
        MOV     OPTFLG,(R5)+            ; Store option flags
        MOV     MSGFLG,(R5)+            ; Store message flags

        MOV     .DEVNM,(R5)+            ; Store device name length
        MOV     .DEVNM+2,(R5)+          ;                   address
        MOV     .NDADR,(R5)+            ; Store destination Ethernet addr
        MOV     .NDADR+2,(R5)+          ;
        MOV     .NDADR+4,(R5)+          ;
```

```
MOVB    .ISAP,(R5)+             ; Store individual SAP
CLRB    (R5)+                   ;   as a word
MOVB    .GSAP,(R5)+             ; Store group SAP
CLRB    (R5)+                   ;   as a word
MOVB    .DSAP,(R5)+             ; Store destination SAP
CLRB    (R5)+                   ;   as a word
MOV     .CTL,(R5)+              ; Store CTL field

MOV     #.SNAP,(R5)+            ; Store SNAP protocol address

FORMAT  #FMTDAT,#PRSERR         ; Format data into ASCII
LOG     #FMTBUF,FMTL            ; Log the text
RETURN

.END    TST802
```

5

DLX Point-to-Point and Multipoint Programming Facilities

The Direct Line Access controller (DLX) gives programs a direct interface to the data link, bypassing the standard DECnet user interface. With DLX, you can communicate with DECnet or non-DECnet based systems. Because DLX does not offer higher-level DECnet services, such as routing and guaranteed delivery, it can give high performance in network applications. DLX also lets you build customized user-level protocols that best suit your applications.

To use DLX, you issue queued input/output (QIO) calls to the NX: device. Your DLX program can communicate with a DLX program on an adjacent DECnet–RSX or non-DECnet node, using the DECnet DDCMP protocol. Your DECnet–RSX node can simultaneously run multiple DECnet and DLX tasks, each possibly communicating with different nodes.

DLX is automatically built for RSX–11M–PLUS systems; it is optional for RSX–11M. It is also optional for RSX–11S systems, but is required for RSX–11S down-line loads and up-line dumps.

5.1 Prerequisites for Tasks Using DLX

Before your system runs a DLX program, the DLX process must be loaded and the circuit set.

The person in charge of network or system management installs the network, usually by executing a command file that contains the command for loading DLX. When DLX is loaded, it resides in the common partition NT.DLX.

The network manager also sets the circuit, either by answering Yes to the NETGEN question that asks about marking the circuit for load, or by issuing the Network Control Program (NCP) SET LINE command. The circuit owner must be DLX. For information on using NCP to set the circuit, refer to the *DECnet–RSX Guide to Network Management Utilities*.

5.2 Writing DLX Programs

DLX programming requires a thorough knowledge of MACRO–11 assembly language and experience in writing real-time application programs.

Since DLX bypasses the higher levels of DECnet, you lose the services at those levels and must therefore include them in your application. Your programs must provide the following:

Flow control	DLX does not support flow control for data transfer. The DLX programs that run on different nodes must therefore synchronize with each other before transferring data. If the tasks are unsynchronized, data can be lost.
Error recovery	The DLX software reports errors, but your program must include error recovery procedures.
Data segmentation	When transmitting data, your program must segment it; the buffer size must be appropriate to the controller devices on the communicating systems. For information on appropriate buffer sizes, consult your network manager.

Note that all incoming and outgoing DLX messages are buffered in a shared network buffer pool. DECnet and other DLX tasks also use these buffers. Depending on the requirements of the tasks sharing the buffers, you may want to increase the size and/or number of buffers to maintain good throughput performance. For information on displaying and setting buffer sizes, refer to the DECnet–RSX network management documentation.

Also note that you must use the /PR:0 switch to task build your DLX programs.

5.2.1 DLX Resources

DLX provides macros and QIOs to use in your application.

The DECnet macro library, NETLIB.MLB, defines the offsets and macros that DLX QIOs use. During NETGEN, this library is transferred to your system. The definition macro DLXDF$ contains definitions for offsets and macros.

Your program must issue .MCALL statements and explicitly invoke the macros, as in the following example:

```
.MCALL DLXDF$  ; extract from macro library
          .
          .
          .
       DLXDF$ ; define DLX symbols
```

DLX QIO functions perform services your application will require. The QIOs for multipoint and point-to-point programming are:

IO.XOP	Open a circuit for your program. This gives your program access to the controller.
IO.XIN	Initialize the circuit after a device error.
IO.XTM	Transmit a message.
IO.XRC	Ready the circuit to receive a message.
IO.XHG	Hang up the circuit without closing it.
IO.XCL	Close the circuit.

5.3 DLX QIOs

DLX requests conform to normal standards for RSX–11 QIOs, including logical unit numbers (LUNs), event flags, I/O status blocks, asynchronous system traps (ASTs), and parameter lists. According to RSX–11 standards, you can use any one of the three macro formats (see Chapter 2). You can use the QIO wait option (QIOW$) to suspend execution of the program until the call completes.

The rest of this chapter describes the DLX QIOs. The descriptions are in the order in which you will probably use the QIOs.

IO.XOP
(Open a Circuit)

5.3.1 IO.XOP — Open a Circuit

Use:

Issue this QIO to open a circuit for DLX transmission and reception. This QIO associates the LUN you specify with the circuit you specify. The circuit is then implicitly initiated, and the DDCMP protocol is started.

Before your application issues the IO.XOP call, the circuit owner must be set to DLX; the circuit must be either ON or in SERVICE state, and the LUN must be assigned to NX:. With devices that implement the DDCMP protocol in software, such as DL11, DUP11, DZ, DHU, and DHV devices, the IO.XOP function does not complete until the task at the other end of the circuit also performs an open or initialize function.

Format:

QIO$ IO.XOP,*lun*,[*efn*],,[*status*],[*ast*],<*p1,p2,p3*>

Arguments:

IO.XOP

is the function code that opens a circuit.

lun

is the logical unit number associated with the circuit.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after the QIO completes.

p1

is the address of an ASCII string that identifies the circuit to open.

The format is:

dev-ctl[*-circuit*][*.tributary*]

where *dev* is the device mnemonic, *ctl* is the decimal value for the controller number, *circuit* is the decimal number of the circuit you are opening, and *tributary* defines the decimal number of the multipoint tributary with which to communicate.

p2

is the length of an ASCII string that identifies the circuit to open.

p3

is a word argument that specifies the timeout value. This value specifies how long to wait to receive a transmitted message. The low-order byte of the word designates the receive timeout value as follows:

timeout = 0 for no receive timer.
timeout = <*n*>

where *n* is the timer value in seconds. The timer value *n* causes the timeout to have a range of *n*–1 to *n*.

Use a zero (0) in the high-order byte of this word.

Completion Status:

IS.SUC (1)	The circuit opened successfully.
177736 IE.ALN (-34.)	The specified LUN is already in use.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177646 IE.NSF (-26.)	Either you identified the circuit incorrectly or it is not in the system.
177760 IE.PRI (-16.)	The circuit you specified is not available for DLX use.
177757 IE.RSU (-17.)	The specified circuit is already in use.

IO.XIN
(Initialize the Circuit)

5.3.2 IO.XIN — Initialize the Circuit

Use:

Issue this QIO to reinitialize a circuit after a fatal device error. When you use this QIO, you must reset the mode and timer values.

With devices that implement the DDCMP protocol in software, such as DL11, DUP11, DZ, DHU, and DHV devices, the IO.XOP function does not complete until the task at the other end of the circuit also performs an open or initialize function.

Format:

QIO$ IO.XIN,*lun*,[*efn*],,[*status*],[*ast*],<*p1*>

Arguments:

IO.XIN

is the function code that initializes the circuit.

lun

is the logical unit number that you assigned when you opened the circuit.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this call completes.

p1

is the timer argument. Use the format for the IO.XOP argument *p3* (Section 5.3.1).

Completion Status:

IS.SUC (1)	The circuit was successfully initialized.
177761 IE.ABO (-15.)	The initialization attempt was aborted. Either a hardware device error occurred, a user issued a hang-up QIO, or the circuit was not hung up.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177733 IE.NLN (-37.)	No open circuit has the specified LUN.

IO.XTM
(Transmit a Message on the Circuit)

5.3.3 IO.XTM — Transmit a Message on the Circuit

Issue this QIO to transmit a message. IO.XTM transfers the data from the buffer whose address and length you specify in *p1* and *p2* to a network buffer for transmission. Before transmitting, you must open the circuit; before transmitting after a device error, you must initialize the circuit.

Format:

QIO$ IO.XTM,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*>

Arguments:

IO.XTM

is the function code for transmitting a message.

lun

is the logical unit number associated with the circuit on which to transmit.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this QIO completes.

p1

is the address of the user buffer that contains the message to transmit.

p2

is the length of the message to transmit, excluding the DDCMP header and checksum.

Completion Status:

IS.SUC (1)	The message was successfully transmitted.
177761 IE.ABO (-15.)	The transmission was aborted because you or the remote user issued a hang-up QIO or because an unrecoverable error occurred in the hardware device. When a message transmission completes with an IE.ABO code, the circuit is hung up. You must either initialize or close and reopen the circuit before using it again.
177775 IE.DNR (-3.)	The hardware device was not ready. The circuit was hung up and not reinitialized.
177776 IE.IFC (-2.)	The LUN is not assigned to NX:.
177733 IE.NLN (-37.)	No open circuit has the specified LUN.
177772 IE.SPC (-6.)	The transmit buffer is too large.

IO.XRC
(Receive a Message on the Circuit)

5.3.4 IO.XRC — Receive a Message on the Circuit

Issue this QIO to receive a message from the remote node. The circuit must already be initialized. You must issue IO.XRC to get any data that a remote node sends. If a remote node sends data, but you have not issued IO.XRC, you get an error report when you next issue this QIO.

Format:

QIO$ IO.XRC,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*>

Arguments:

IO.XRC

is the function code for receiving a message.

lun

is the logical unit number associated with the circuit on which to receive the message.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this call completes.

p1

is the address of the user buffer to receive the message.

p2

is the length in bytes to allocate for the receive buffer. The length of the received message cannot exceed the size of the system buffer, regardless of the length that you specify for *p2*.

Completion Status:

IS.SUC (1)	You successfully received a message from the remote node. The second word of the I/O status block contains the number of bytes you received.
177761 IE.ABO (–15.)	The receive function was aborted. Either you or the remote user issued a hang-up QIO, or an unrecoverable hardware device error occurred. When a receive is aborted, the circuit is hung up. You must either initialize or close and reopen the circuit before using it again.
177763 IE.DAO (–13.)	Either a message was received before a receive QIO was issued and the data was lost or the user buffer was too small, and the message was truncated. The user buffer length is in the second word of the I/O status block.
177775 IE.DNR (–3.)	The hardware device was not ready. The circuit was hung up and not reinitialized.
177776 IE.IFC (–2.)	The LUN is not assigned to NX:.
177733 IE.NLN (–37.)	No open circuit has the specified logical unit number.
177641 IE.TMO (–95.)	A timeout condition occurred. No message arrived within the timer interval that you specified when you opened or initialized the circuit.

177774 IE.VER (-4.) An error occurred on the circuit. The second word of the I/O status block contains the error code. The error codes are as follows:

100361	DDCMP transmit error threshold exceeded
100362	Operation aborted
100363	Message received without receive pending
100364	Start received
100366	Circuit physically disconnected
100370	General error
100374	DDCMP reply timeout threshold exceeded
100376	DDCMP receive error threshold exceeded

IO.XHG
(Hang Up the Circuit)

5.3.5 IO.XHG — Hang Up the Circuit

Use:

This QIO stops operations on a circuit. IO.XHG does not close a circuit, but to resume operations, you must either initialize or close and reopen the circuit.

Format:

QIO$ IO.XHG,*lun*,[*efn*],,[*status*],[*ast*]

Arguments:

IO.XHG

is the function code that hangs up the circuit.

lun

is the logical unit number associated with the circuit.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this call completes.

Completion Status:

IS.SUC (1)	This circuit was hung up.
177776 IE.IFC (–2.)	The LUN is not assigned to NX:.
177733 IE.NLN (–37.)	No open circuit has the specified LUN.

IO.XCL
(Close the Circuit)

5.3.6 IO.XCL — Close the Circuit

Use:

Issue the IO.XCL call to close an open circuit and stop the DDCMP protocol. If you have a dial-up connection, the circuit will hang up only after the close completes.

Format:

QIO$ IO.XCL,*lun*,[*efn*],,[*status*],[*ast*]

Arguments:

IO.XCL

is the function code that closes the circuit.

lun

is the logical unit number associated with the circuit.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block. On completion, the block contains the QIO completion status in the low-order byte of the first word (see under "Completion Status").

ast

is the entry point into an optional user-written AST routine to execute after this call completes.

Completion Status:

IS.SUC (1)	The circuit was successfully closed.
177776 IE.IFC (–2.)	The LUN is not assigned to NX:.
177733 IE.NLN (–37.)	No open circuit has the specified LUN.

5.3.7 Programming Examples

The following two programs use DLX to send and receive data. These examples are also included in your tape or disk kit.

5.3.7.1 Transmit Example

The XTS program reads data from a user or an indirect command file and transmits the data to the cooperating XTR program on a remote node.

```
        .TITLE  XTS - DLX TRANSMITTER
        .IDENT  /V01.01/

;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
; The XTS program transmits data across an "error free" circuit to a
; receiver task.  The data can be read in from a user at a terminal or
; or from an indirect command file. The receiver task, XTR, echoes the
; received data back over the circuit.
;
; You must run this program on a system that supports write break-throughs.
;
; To assemble, use the following command string:
;
;       MAC XTS,XTS/-SP/LI:TTM=IN:[130,10]NETLIB/ML,IN:[200,200]XTS
;
; To task build, use the following command string:
;
;       XTS/PR:0,XTS/-SP=XTS,IN:[130,10]NETLIB/LB:GCL
;       /
;       STACK=30
;       UNITS=4
;       ASG=TI:1:2:3:4
;       TASK=...XTS
;       //
;
; Note: The IN: device must be the DECnet distribution device
;       after the PREGEN procedure (if any) has been performed.

; The following is an example of the XTS-XTR dialog:
;
;       >XTS
;       LINE: DMC-0
;       XTS>THIS IS A TEST OF XTS-XTR
;       THIS IS A TEST OF XTS-XTR
;
```

(continued on next page)

```
;       XTS>TESTING
;       TESTING
;
;       XTS>^Z
;       >
;
; XTR must be running on the remote system in order to receive the message
; from XTS and return an echo.
;-
        .SBTTL  LOCAL MACROS

        .MACRO  EPRINT  ERRMSG
        MOV     #ERRMSG,R0
        CALL    $EPRINT
        .ENDM   EPRINT

        .SBTTL  MACRO CALLS

        .MCALL  QIOW$,QIO$,QIOW$S,ALUN$S,EXIT$S,EXST$S,FSRSZ$,ASTX$S
        .MCALL  GCL$,GCLDF$,CALLR,DLXDF$

        DLXDF$                          ;Define DLX function codes

        .SBTTL  CONSTANTS
;
; LUN assignments:
;
        TILUN=1                         ;LUN for TI
        CHNLUN=2                        ;LUN for error free circuit
        ERRLUN=3                        ;LUN for errors
        CMDLUN=4                        ;LUN for command lines
;
; Event flag assignments:
;
        TIEFN=1                         ;Event flag for terminal I/O
        CHNEFN=2                        ;Event flag for circuit
        ERREFN=3                        ;Event flag for error messages
        CMDEFN=4                        ;Event flag for command lines

        .SBTTL  DATA
;
; Define GCL parameters
;
        GCLDF$  CMDLUN,CMDEFN,<XTS>,CMDBUF,80.
;
; Define FSR size
;
        FSRSZ$  1                       ;Room for 1 file (GCL)
;****
; DPBs
;****
WRITE:  QIOW$   IO.WVB,TILUN,TIEFN,,,,<0,0,40>

ERDPB:  QIOW$   IO.WVB,ERRLUN,ERREFN,,,,<0,0,40>


REC1:   QIO$    IO.XRC,CHNLUN,,,R1SB,RECAST,<R1BUF,80.>
REC2:   QIO$    IO.XRC,CHNLUN,,,R2SB,RECAST,<R2BUF,80.>
```

```
CLOSE:  QIOW$   IO.XCL,CHNLUN,CHNEFN
;
; Exit-with-status word
;
EXSTAT: .BLKW   1                               ;Exit status
;
; Circuit I/O status block
;
CHNSB:  .BLKW   2
;
; AST saved I/O status block
;
IOSB:   .BLKW   1
;
; Circuit receive I/O status blocks
;
R1SB:   .BLKW   2                       ;Status of first receive
        .WORD   R1BUF                   ;Address of buffer
        .WORD   HNGRC1                  ;Address of receive posting routine

R2SB:   .BLKW   2                       ;Status of second receive
        .WORD   R2BUF                   ;Address of buffer
        .WORD   HNGRC2                  ;Address of receive posting routine
;
; Buffer for command line
;
CMDBUF: .BLKB   82.
        .EVEN
;
; Circuit receive buffers
;
R1BUF:  .BLKB   80.
R2BUF:  .BLKB   80.
        .EVEN
;****
; TEXT STRINGS:
;****
;
; Header for error messages
;
XTSEM:  .ASCIZ  /XTS -- /
;
; Temporary prompt
;
PROMPT: .ASCIZ  <15><12>/LINE: /
;
; Error messages
;
        .ENABL  LC
        .NLIST  BEX
GCLERR: .ASCIZ  /Command line read error/
NSFERR: .ASCIZ  /No such command file/
DLXERR: .ASCIZ  /DLX not loaded/
OPNERR: .ASCII  /Unable to open line -- /
BUFOPN: .BLKB   7
XMTERR: .ASCII  /Error transmitting data -- /
BUFXMT: .BLKB   7
RECERR: .ASCII  /Error receiving data -- /
```

(continued on next page)

```
BUFREC: .BLKB   7
        .LIST   BEX
        .EVEN

        .SBTTL  XTS - XTS MAIN LINE
;+
; XTS -- Main line of XTS code
;-

XTSEP::

        MOV     #EX$SUC,EXSTAT          ;Assume exit with status
;
; Assign LUN to circuit
;
        ALUN$S  #CHNLUN,#"NX,#0
        BCC     10$                     ;If CC, all okay
        EPRINT  DLXERR                  ;Else, assume DLX not loaded
        BR      EXIT                    ;and leave
;
; Prompt user for line ID
;
10$:    MOV     $CLPMT,-(SP)            ;Save current prompt
        MOV     #PROMPT,$CLPMT          ;Prompt string
        CALL    GCL                     ;Get a command line
        MOV     (SP)+,$CLPMT            ;Restore prompt
        BCS     EXIT                    ;If CS, assume EOF
        TST     R5                      ;Blank line?
        BEQ     10$                     ;If EQ, yes - try again
;
; Open access to the line
;
        QIOW$S  #IO.XOP,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5>
        BCS     15$                     ;IF CS, error
        MOVB    CHNSB,R1                ;Successful?
        BPL     20$                     ;If PL, yes
        MOV     #BUFOPN,R0              ;Else, get buffer address
        CLR     R2                      ;Zero suppression
        CALL    $CBOMG                  ;Convert number
        CLRB    (R0)                    ;Make string ASCIZ
15$:    EPRINT  OPNERR                  ;Open error
        BR      EXIT
;
; Hang an asynchronous read on line
;
20$:    CALL    HNGRC1
        CALL    HNGRC2
        BCS     EXIT                    ;If CS, error
;
; Get command line
;
30$:    CALL    GCL                     ;Get command line
        BCS     EXIT                    ;If CS, assume EOF
        TST     R5                      ;Empty line?
        BEQ     30$                     ;If EQ, yes - try again
;
; Transmit the buffer
;
        CALL    XMIT                    ;Transmit the buffer
        BCC 30$                         ;If CC, get next message
;
```

```
; Close the line
;
EXIT:   DIR$    #CLOSE
;
; Exit XTS
;
        EXST$S  EXSTAT                  ;Try to exit-with-status
        EXIT$S                          ;Else, just exit

        .SBTTL  GCL - GET COMMAND LINE
;+
; **-GCL-Get command line
;
; This routine reads a command line for XTS. The input can be from
; TI: or from an indirect command file. Return with carry set for error
; or EOF.
;
; Inputs:
;       None
;
; Outputs:
;       R4=address of command line
;       R5=size of command line in bytes
;       Carry bit set/cleared
;
; Effects:
;       R4,R5 modified
;-
GCL:    GCL$                            ;Get command line
        MOV     $CLIOS,R5               ;Point to I/O status block
        TSTB    (R5)                    ;Error?
        BGT     40$                     ;If GT, no

        CMPB    #IE.EOF,(R5)            ;End of file?
        BEQ     30$                     ;If EQ, yes - set C and return

        CMPB    #IE.ABO,(R5)            ;Was read killed by receive?
        BEQ     30$                     ;If EQ, yes - return with C-SET

        CMPB    #IE.NSF,(R5)            ;No such file error?
        BNE     10$                     ;If NE, no
        EPRINT  NSFERR                  ;Else, say so
        CALL    ECHO                    ;Echo command line

        CLR     R5                      ;Set command line length to 0
        BR      50$                     ;and return empty

10$:    EPRINT  GCLERR                  ;Print get command line error
20$:    TSTB    $CLEVL                  ;Terminal input?
        BNE     30$                     ;If NE, no - fatal error
        BR      GCL                     ;Else, prompt again
30$:    SEC                             ;Set carry
        BR      50$                     ;and exit
;
; Get size and address of command line
;
40$:    MOV     $CLBUF,R4               ;Get address of command line
        MOV     2(R5),R5                ;Get size of command line
        CLC                             ;Set success
```

(continued on next page)

```
50$:
        RETURN                          ;Return

        .SBTTL  HNGRC1 - HANG ASYNCHRONOUS READ ON LINE
;+
; **-HNGRC1 - Hang an asynchronous read on the circuit
; **-HNGRC2 -
;
; Inputs:
;       None
;
; Outputs:
;       Receive hung on line
;
;-
        .ENABL  LSB
HNGRC1:
        CALL    $SAVAL                  ;Save all registers
        DIR$    #REC1                   ;Hang receive
        BCS     10$                     ;If CS, error
        BR      20$                     ;and continue in common code

HNGRC2:
        CALL    $SAVAL                  ;Save all registers
        DIR$    #REC2                   ;Hang receive
        BCC     20$                     ;If CC, success
10$:    EPRINT  RECERR                  ;Receive error
        SEC                             ;Indicate failure
20$:    RETURN                          ;Return
        .DSABL  LSB

        .SBTTL  XMIT - TRANSMIT DATA OVER LINE
;+
; **-XMIT - Transmit data over line
;
; Inputs:
;       R4 = Address of data
;       R5 = Length of data
;

; Outputs:
;       Data transmitted
;-
XMIT:
        QIOW$S  #IO.XMT,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5>
        BCS     10$                     ;If CS, error
        MOVB    CHNSB,R1                ;Successful?
        BPL     20$                     ;If PL, yes
        MOV     #BUFXMT,R0              ;Else, get buffer address
        CLR     R2                      ;Zero suppression
        CALL    $CBOMG                  ;Convert number
        CLRB    (R0)                    ;Make string ASCIZ
10$:    EPRINT  XMTERR                  ;Transmit error
        SEC                             ;Indicate failure
20$:    RETURN

        .SBTTL  RECAST - AST FOR CHANNEL READ COMPLETE
;+
; **-RECAST - AST for circuit read complete
;
; Inputs:
;       (SP) = Address of I/O status block
;
```

```
; Outputs:
;       1. Another read hung on channel (if last receive succeeded)
;       2. Buffer read from channel is echoed on terminal
;-
RECAST:
        MOV     (SP),IOSB               ;Save I/O status block address
        MOV     R1,(SP)                 ;Save R1
        MOV     IOSB,R1                 ;Get I/O status block address
        TSTB    (R1)                    ;Successful completion?
        BPL     10$                     ;If PL, yes - write it out
        CALLR   EXIT                    ;Else, close line and exit
10$:    MOV     2(R1),WRITE+Q.IOPL+2    ;Set length of buffer to write
        MOV     4(R1),WRITE+Q.IOPL      ;Set buffer address
        DIR$    #WRITE                  ;Write buffer to terminal
        CALL    @6(R1)                  ;Hang another receive
        MOV     (SP)+,R1                ;Restore R1
        ASTX$S

        .SBTTL  $EPRINT -- PRINT ERROR MESSAGE
;+
; **-$EPRINT- Print error message
;
; Prints the specified error message prefixed by "XTS -- ".
; Sets the exit-status as "EX$ERR".
;
; Inputs:
;       R0=Address of message
;
; Outputs:
;       Error message printed on TI:
;       EXSTAT = EX$ERR
;
; Effects:
;       No registers modified
;-
        .ENABL  LSB
$EPRINT:
        MOV     R0,-(SP)                ;Save R0
        MOV     #EX$ERR,EXSTAT          ;Set exit status to "ERROR"
        MOV     #44,ERDPB+Q.IOPL+4      ;Set vertical format to prompt
        MOV     #XTSEM,R0               ;Get prefix message
        CALL    5$                      ;Print prefix
        MOV     #53,ERDPB+Q.IOPL+4      ;Set vert. format to overprint
        MOV     (SP)+,R0                ;Get address of message
PRINT2:
5$:     MOV     R0,ERDPB+Q.IOPL         ;Set address of message
10$:    TSTB    (R0)+                   ;Null byte?
        BNE     10$                     ;If NE, no - keep looking
        DEC     R0                      ;Don't count null
        SUB     ERDPB+Q.IOPL,R0         ;Calculate length of string
        MOV     R0,ERDPB+Q.IOPL+2       ;Set length of string
        DIR$    #ERDPB                  ;Issue directive
        MOV     #40,ERDPB+Q.IOPL+4      ;Restore vertical format to normal
        RETURN
        .DSABL  LSB

        .SBTTL  ECHO - ECHO COMMAND LINE
```

(continued on next page)

```
;+
; **-ECHO-Echo command line
;
; This routine echoes the current command line if it came from an indirect
; command file.
;
; Inputs:
;       $CLEVL=Indicates command file level
;       $CLBUF=Pointer to start of ASCIZ command line
;
; Outputs:
;       LINE FEED appended to command line and command line echoed on TI:
;
; Effects:
;       R0, R1 modified
;-
ECHO:
        TSTB    $CLEVL                  ;Command from terminal?
        BEQ     10$                     ;If EQ, yes - don't echo
        MOV     $CLBUF,R0               ;Point to command line
        CALL    PRINT2                  ;Print line on error LUN
10$:    RETURN

        .END    XTSEP
```

5.3.7.2 Receive Example

The XTR program uses DLX QIOs to receive data from the cooperating XTS task on a remote node.

```
        .TITLE  XTR - DLX RECEIVER
        .IDENT  /V01.01/
;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
; This software is furnished under a license and may be used and copied
; only in accordance with the terms of such license and with the
; inclusion of the above copyright notice. This software or any other
; copies thereof may not be provided or otherwise made available to any
; other person. No title to and ownership of the software is hereby
; transferred.
;
; The information in this software is subject to change without notice
; and should not be construed as a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or reliability of its
; software on equipment which is not supplied by Digital.
;
;
; The XTR program receives and echoes data over a circuit.  Use it in
; conjunction with the XTS program.
;
; To assemble, use the following command string:
;
;       MAC XTR,XTR/-SP=IN:[130,10]NETLIB/ML,IN:[200,200]XTR
;
; To task build, use the following command string:
;
;       XTR/PR:0,XTR/-SP=XTR,IN:[130,10]NETLIB/LB:GCL
;       /
;       STACK=30
;       UNITS=3
;       ASG=TI:1:2:3
;       TASK=...XTR
;       //
;
;    Note: The IN: device must be the DECnet distribution device
;          after the PREGEN procedure (if any) has been performed.
;
; The following is an example of the XTS-XTR dialog:
;
;       >XTS
;       LINE: DMC-0
;       XTS>THIS IS A TEST OF XTS-XTR
;       THIS IS A TEST OF XTS-XTR
;
;       XTS>TESTING
;       TESTING
;
;       XTS>^Z
;       >
;
```

(continued on next page)

```
; Start XTR before starting the remote XTS program.  Start XTR as follows:
;
;       >XTR
;       LINE: DUP-0
;
; When you are finished with these programs, abort XTR.
;-
        .SBTTL  LOCAL MACROS

        .MACRO  EPRINT  ERRMSG
        MOV     #ERRMSG,R0
        CALL    $EPRINT
        .ENDM   EPRINT

        .SBTTL  MACRO CALLS

        .MCALL  QIOW$,QIO$,QIOW$S,ALUN$S,EXIT$S,EXST$S,ASTX$S,WTSE$S
        .MCALL  GCL$,GCLDF$,DLXDF$,DLXBUF

        DLXDF$                          :Define DLX function codes and overhead

        .SBTTL  CONSTANTS
;
; Receive buffer size
;
        BUFSIZ = 90.
;
; LUN assignments:
;
        TILUN=1                         ;LUN for TI
        CHNLUN=2                        ;LUN for error-free circuit
        ERRLUN=3                        ;LUN for errors
;
; Event flag assignments:
;
        TIEFN=1                         ;Event flag for terminal I/O
        CHNEFN=2                        ;Event flag for circuit
        ERREFN=3                        ;Event flag for error messages
        DONE=4                          ;Event flag signaling completion

        .SBTTL  DATA

;
; Define GCL parameters
;
        GCLDF$  TILUN,TIEFN,<LINE>,R1BUF,BUFSIZ

;****
; DPBs
;****

ERDPB:  QIOW$   IO.WVB,ERRLUN,ERREFN,,,,<0,0,40>

REC1:   QIO$    IO.XRC,CHNLUN,,,R1SB,RECAST,<R1BUF,BUFSIZ>
REC2:   QIO$    IO.XRC,CHNLUN,,,R2SB,RECAST,<R2BUF,BUFSIZ>
```

```
XMT:    QIOW$   IO.XTM,CHNLUN,CHNEFN,,CHNSB,,<0,0>

START:  QIOW$   IO.XIN,CHNLUN,CHNEFN,,CHNSB

CLOSE:  QIOW$   IO.XCL,CHNLUN,CHNEFN

;
; Circuit I/O status block
;
CHNSB:  .BLKW   2
;
; Temporary location to contain IOSB address
;
IOSB:   .BLKW   1
TEMP:   .BLKW   1

;
; Circuit receive I/O status blocks
;
R1SB:   .BLKW   2                       ;Status of first receive
        .WORD   R1BUF                   ;Address of buffer
        .WORD   HNGRC1                  ;Address of receive posting routine

R2SB:   .BLKW   2                       ;Status of second receive
        .WORD   R2BUF                   ;Address of buffer
        .WORD   HNGRC2                  ;Address of receive posting routine

;
; Circuit receive buffers
;
        DLXBUF  R1BUF,BUFSIZ            ;First buffer descriptor
        DLXBUF  R2BUF,BUFSIZ            ;Second buffer descriptor
        .EVEN

;****
; Text strings:
;****

;
; Header for error messages
;
XTREM:  .ASCIZ  /XTR -- /

;
; Error messages
;
        .ENABL  LC
        .NLIST  BEX
GCLERR: .ASCIZ  /Command line read error/
DLXERR: .ASCIZ  /DLX not loaded/
OPNERR: .ASCII  /Unable to open line -- /
BUFOPN: .BLKB   7
XMTERR: .ASCII  /Error transmitting data -- /
BUFXMT: .BLKB   7
```

(continued on next page)

```
RECERR: .ASCII  /Error receiving data -- /
BUFREC: .BLKB   7
        .LIST   BEX
        .EVEN

        .SBTTL  XTREP - XTR MAIN LINE
;+
; XTREP -- Main line of XTR code
;
; Prompt user for line to open and loop all received messages over the same line
;
; Inputs:
;       None.
;
; Outputs:
;       Loop all messages indefinitely.
;-

XTREP::
        CLR     R3
;
; Assign LUN to circuit
;
        ALUN$S  #CHNLUN,#"NX,#0
        BCC     10$                     ;If CC, all okay
        EPRINT  DLXERR                  ;Else, assume DLX not loaded
        BR      99$                     ;and leave
;
; Prompt user for line ID
;
10$:    CALL    GCL                     ;Get a command line
        BCS     99$                     ;If CS, assume EOF
        TST     R5                      ;Blank line?
        BEQ     10$                     ;If EQ, yes - try again
;
; Open access to the line
;
        QIOW$S  #IO.XOP,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5>
        BCS     15$                     ;If CS, error
        MOVB    CHNSB,R1                ;Successful?
        BPL     20$                     ;If PL, yes
        MOV     #BUFOPN,R0              ;Else, get buffer address
        CLR     R2                      ;Zero suppression
        CALL    $CBOMG                  ;Convert number
        CLRB    (R0)                    ;Make string ASCIZ
15$:    EPRINT  OPNERR                  ;Open error
        BR      99$
;
; Hang an asynchronous read on line
;
20$:    CALL    HNGRC1
        BCS     99$                     ;If CS, error

        CALL    HNGRC2                  ;Hang second receive
        BCS     99$                     ;If CS, error
```

```
;
; The rest is AST-driven. Pretend we are waiting for something.
;
        WTSE$S  #DONE                   ;Wait for completion (never happens)

99$:    DIR$    #CLOSE                  ;Close down the line
        EXIT$S                          ;Exit

        .SBTTL  GCL - GET COMMAND LINE
;+
; **-GCL-Get command line
;
; This routine reads a command line for XTR.  The input can be from TI:
; or from an indirect command file. Return with carry bit set for error or EOF.
;
; Inputs:
;       NONE
;
; Outputs:
;       R4=ADDRESS OF COMMAND LINE
;       R5=SIZE OF COMMAND LINE IN BYTES
;       C-BIT SET/CLEARED
;
; Effects:
;       R4,R5 MODIFIED.
;-

GCL:    GCL$                            ;Get command line
        MOV     $CLIOS,R5               ;Point to I/O status block
        TSTB    (R5)                    ;Error?
        BGT     40$                     ;If GT, no

        CMPB    #IE.EOF,(R5)            ;End of file?
        BEQ     30$                     ;If EQ, yes - set carry and return

        CMPB    #IE.ABO,(R5)            ;Was read killed by receive?
        BEQ     30$                     ;If EQ, yes - return with carry set

10$:    EPRINT  GCLERR                  ;Print get command line error
20$:    TSTB    $CLEVL                  ;Terminal input?
        BNE     30$                     ;If NE, no - fatal error
        BR      GCL                     ;Else,  re-prompt
30$:    SEC                             ;Set carry
        BR      50$                     ;and exit
;
; Get size and address of command line
;
40$:    MOV     $CLBUF,R4               ;Get address of command line
        MOV     2(R5),R5                ;Get size of command line
        CLC                             ;Set success

50$:
        RETURN                          ;Global return

        .SBTTL  HNGRC1 - HANG ASYNCHRONOUS READ ON LINE
```

(continued on next page)

```
        .SBTTL  HNGRC2 - HANG SECOND ASYNCHRONOUS READ
;+
; **-HNGREC - Hang an asynchronous read on the circuit
; **-HNGRC2 - Hang second asynchronous read on circuit
;
; Inputs:
;       None.
;
; Outputs:
;       Receive hung on line
;-
        .ENABL  LSB
HNGRC1:
        DIR$    #REC1                   ;Hang read
        BCS     10$                     ;If cs, error
        BR      20$                     ;And continue in common code
HNGRC2:
        DIR$    #REC2                   ;Hang read
        BCC     20$                     ;If CC return
10$:    EPRINT  RECERR                  ;Receive error
        SEC                             ;Indicate failure
20$:    RETURN
        .DSABL  LSB

        .SBTTL  XMIT - TRANSMIT DATA OVER LINE
;+
; **-XMIT - Transmit data over line
;
; Inputs:
;       None
;
; Outputs:
;       Data transmitted
;-
XMIT:
        MOV     R1,-(SP)                ;Save R1
        DIR$    #XMT                    ;Transmit data
        BCS     10$                     ;If CS, error
        MOVB    CHNSB,R1                ;Successful ?
        BPL     20$                     ;If PL, yes
        MOV     #BUFXMT,R0              ;Else, get buffer address
        CLR     R2                      ;Zero suppression
        CALL    $CBOMG                  ;Convert number
        CLRB    (R0)                    ;Make string ASCIZ
10$:    EPRINT  XMTERR                  ;Transmit error
        SEC                             ;Indicate failure
20$:    MOV     (SP)+,R1                ;Restore R1
        RETURN
        .DSABL  LSB

        .SBTTL  RECAST - AST FOR CIRCUIT READ COMPLETE
;+
; **-RECAST - AST for circuit read complete
;
; Inputs:
```

```
;       (SP) = ADDRESS OF I/O STATUS BLOCK
;
; Outputs:
;       1. Another read hung on circuit
;       2. Buffer read from circuit is echoed over line
;-
RECAST:
        MOV     (SP),TEMP               ;Save IOSB address
        MOV     R1,(SP)                 ;Save R1
        MOV     TEMP,R1                 ;R1 -> IOSB
        TSTB    (R1)                    ;Successful completion?
        BPL     10$                     ;If PL, yes - transmit message
        TST     R3                      ;Been through this code last time?
        BNE     20$                     ;Yes - post receive and return
        INC     R3                      ;Mark
        DIR$    #START                  ;Else, restart the line
        BCC     5$                      ;If success, continue
        IOT                             ;Else abort
5$:     TSTB    CHNSB                   ;Success?
        BPL     20$                     ;Yes - continue
        IOT                             ;Else fatal error - abort
10$:    CLR     R3                      ;Clear flag
        MOV     2(R1),XMT+Q.IOPL+2      ;Set length of buffer to transmit
        BEQ     20$                     ;If EQ, no buffer to transmit?
        MOV     4(R1),XMT+Q.IOPL        ;Set address of buffer
        CALL    XMIT                    ;Echo message back over line
20$:    CALL    @6(R1)                  ;Hang another receive on circuit
                                        ;Ignore any errors
        MOV     (SP)+,R1                ;Restore R1
        ASTX$S                          ;Exit AST

        .SBTTL  $EPRINT -- PRINT ERROR MESSAGE
;+
; **-$EPRINT-Print error message
;
; Prints the specified error message, with an "XTR -- " prefix.
; Sets the exit-status as "EX$ERR".
;
; Inputs:
;       R0=Address of message
;
; Outputs:
;       Error message printed on TI:
;
; Effects:
;       No registers modified
;-
$EPRINT:
        MOV     R0,-(SP)                ;Save R0
        MOV     #44,ERDPB+Q.IOPL+4      ;Set vertical format to prompt
        MOV     #XTREM,R0               ;Get prefix message
        CALL    5$                      ;Print prefix
        MOV     #53,ERDPB+Q.IOPL+4      ;Set vert. format to overprint
        MOV     (SP)+,R0                ;Get address of message
5$:     MOV     R0,ERDPB+Q.IOPL         ;Set address of message
```

(continued on next page)

```
10$:    TSTB    (R0)+                   ;Null byte?
        BNE     10$                     ;If NE, no - keep looking
        DEC     R0                      ;Don't count null
        SUB     ERDPB+Q.IOPL,R0         ;Calculate length of string
        MOV     R0,ERDPB+Q.IOPL+2       ;Set length of string
        DIR$    #ERDPB                  ;Issue directive
        MOV     #40,ERDPB+Q.IOPL+4      ;Restore vertical format to normal
        RETURN

        .END    XTREP
```

6

LAT Programming Facilities

The Local Area Transport (LAT) communications protocol runs on terminal servers and host systems. It handles communications among local area network (LAN) devices attached to an Ethernet. Terminal servers are communications servers to which a number of devices, such as user terminals and printers, are attached. An application on a host node on the Ethernet can make logical connections to these devices through the terminal server. During communication between the host node and remote device, the terminal server is almost transparent.

LAT software is available with DECnet–RSX–11M–PLUS and DECnet–Micro/RSX. The LAT protocol and DECnet protocol are different, but they coexist on the same Ethernet.

This chapter is written for programmers who are experienced in using RSX programming directives to write applications for directly-attached devices. The chapter describes how to write new applications or modify applications when the target device is not attached to the host, but to a remote terminal server elsewhere on the Ethernet. The chapter supplements the full-duplex terminal driver information in the *RSX–11M/M–PLUS I/O Drivers Reference Manual* or the *Micro/RSX I/O Drivers Reference Manual.*

Throughout the chapter, "device" refers to a hardware terminal device. "Terminal" refers to the operating system's data structures for handling the hardware device.

The chapter covers the following topics:

- Components of the LAT environment
- Programming steps for applications to LAT terminals

- Directives for programming LAT terminals

For a thorough introduction to the LAT environment, refer to the *Local Area Transport (LAT) Network Concepts* manual.

6.1 Components of the LAT Environment

The LAT environment consists of components on the terminal server and host, along with the Ethernet network that lets them communicate. Figure 6–1 illustrates an application using a LAT connection.

Figure 6–1: Using a LAT Connection

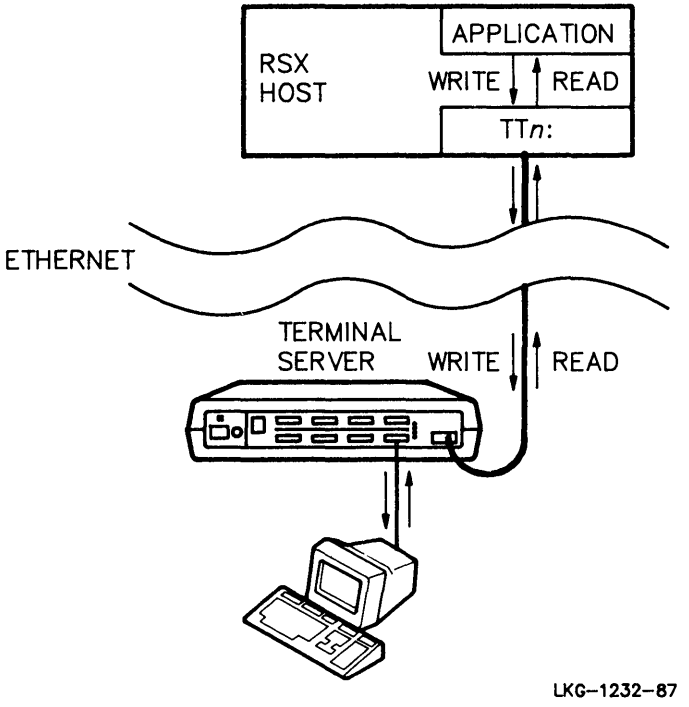
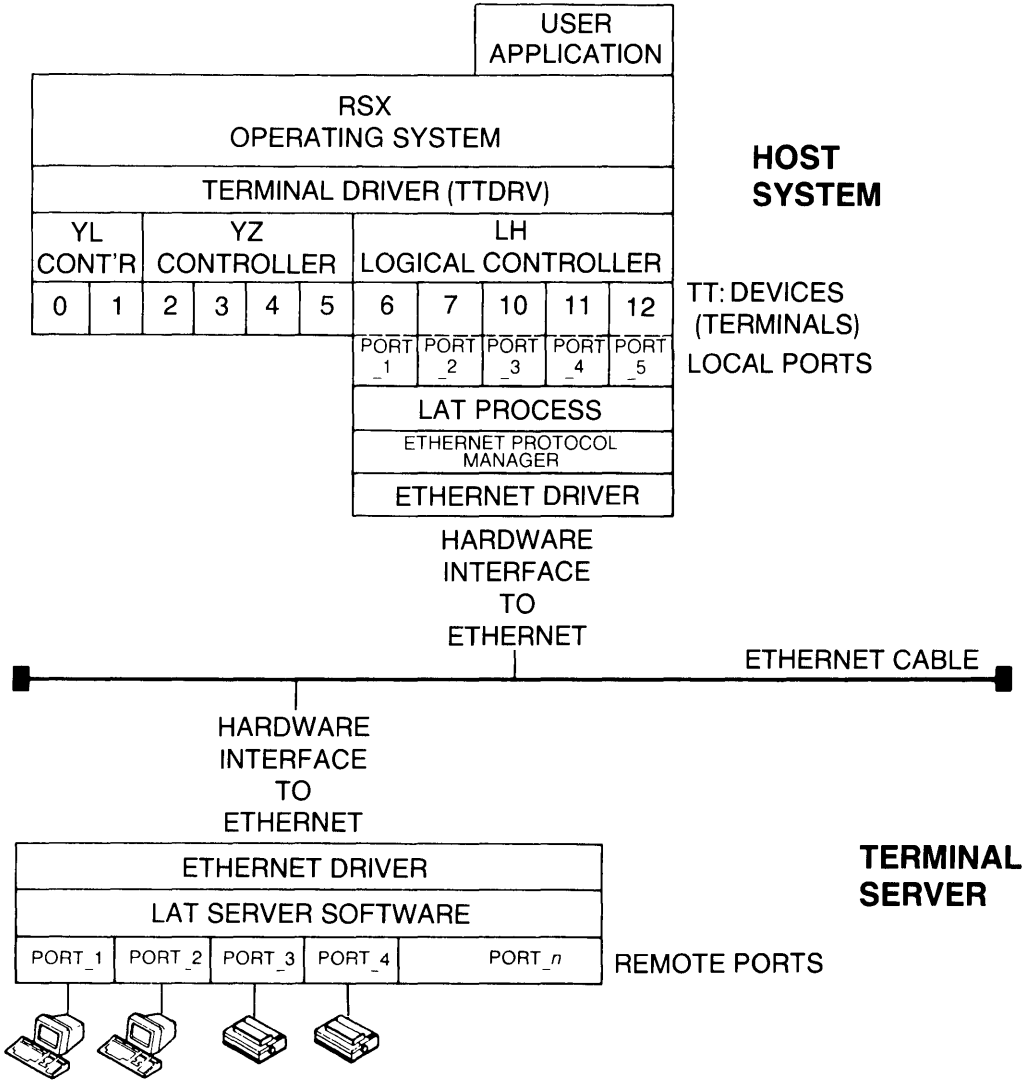


Figure 6–2 gives a more detailed picture of the components that make it possible for applications to communicate with remote LAT devices.

Figure 6–2: LAT Components for Applications



LKG–1029–87

The terminal server is a piece of hardware that has a physical connection to the Ethernet. It also has connections to devices such as user terminals, printers, and so on. A device attached to the terminal server is the target of your application.

Note that LAT software exists on both the host and terminal server. The LAT protocol is implemented on the host by the LAT process and on the terminal server by LAT server software. The LAT process on the host provides an interface between the terminal driver and Ethernet driver. It sends and receives messages having the LAT protocol.

The LAT Control Program (LCP) is a network management interface to the LAT process and operating system. This chapter assumes that someone (called a "network manager") has responsibility for using LCP to set up the host LAT environment. At your site, a system manager or programmer may perform the network manager function. The *DECnet–RSX Guide to Network Management Utilities* includes information on LCP.

When a user and provider of LAT resources have a logical connection, a session exists. Applications can initiate sessions from the host to the terminal server, and interactive users can initiate sessions from the terminal server to the host.

Each end point of the LAT session is a port. The figure shows ports on the RSX host that relate to local terminals (TT: devices) and ports on the terminal server to which user terminals and printers are attached. In order to perform I/O operations on the device at the remote port, your application must make a connection between the local (host) port and the remote (terminal server) port. The next sections describe the local port and remote port.

6.1.1 The Local Port

The local port on the RSX host is a LAT terminal. To create ports, the network manager uses LCP and creates a number of LAT terminals. LAT terminals are TT: devices for use only in LAT sessions. While other TT: devices perform I/O operations to attached devices, the LAT terminals send and receive I/O across a network connection. Unlike other TT: devices, which are created by the SYSGEN procedure, LAT terminals are created after the SYSGEN and NETGEN procedures are complete. The numbering of the LAT terminals starts with the first available number after the numbers for hard-wired terminals, in octal notation. For example, if the system already has 7 terminals with the numbers zero (0) through 6, creating 3 new LAT terminals creates TT7:, TT10:, and TT11:. These terminals would have local port names of PORT__1, PORT__2, and PORT__3.

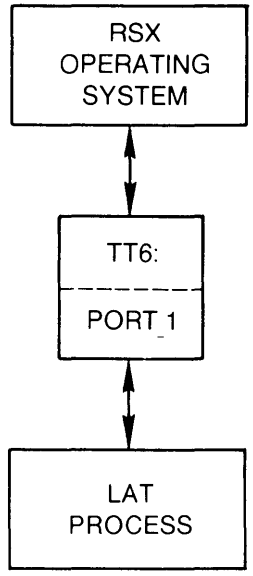
A LAT terminal can be an interactive or application terminal. Interactive terminals are those that remote interactive users use to log on to the host. Application terminals are those that local applications use to connect to a remote device.

LCP normally creates LAT terminals as interactive terminals. Application terminals can be created in several ways. When initially creating LAT terminals, the network manager can specify a number to reserve for application use. Alternatively, after the LAT terminals exist, the network manager or an application can change an interactive terminal into an application terminal by specifying the target information for the terminal. The target information comprises the name of a terminal server and a port and/or service on that terminal server to which any connect request that follows will be directed.

To specify the target information and change an interactive terminal into an application terminal, the network manager issues the LCP SET PORT command. An application does this by issuing the Set Multiple Characteristics (SF.SMC) programming directive, including the appropriate characteristics (Section 6.3.4).

Although the operating system communicates with a TT: device and the LAT process with a port, both refer to the same logical device, as Figure 6–3 illustrates. Figure 6–3 uses as examples the same TT: and port numbers as in Figure 6–2.

Figure 6–3: The LAT Terminal and Local Port



LKG–1034–87

The MCR command CON DISPLAY FOR *LH lists the LAT terminals on your host as TT: devices associated with the LH logical controller, as in the following example:

```
TT11: LHA0:
TT12: LHA1:
TT13: LHA2:
```

The LCP SHOW PORT command lists your host's application terminals with the target information for each. It also lists the interactive terminals currently in use.

6.1.2 The Remote Port

The target for your application is a device at a remote port. The remote port is a physical outlet on the terminal server. Each terminal server has multiple ports. Input- or output-only devices, such as printers or badge readers, or standard input/output user terminals can be attached to terminal server ports. This chapter assumes that someone is responsible for the terminal server, and calls this person the "server manager."

Terminal server ports have names. They can be default names or names that the terminal server manager created. To specify the target for your application, you can always choose to specify the name of the port to which the device is attached.

On some terminal servers, the server manager can also designate a service name for one or more ports. A service is a resource such as a printer, card reader, or computer, that the terminal server makes available to network users. For example, Port_3 and Port_4 of the terminal server in Figure 6–2 could have the service name PRINTER. If the server manager assigns service names, an application can specify the target by its service name instead of its port. When the terminal server receives a request for a service, it passes the request to any free port that offers the service. This frees the application from dependence on a particular configuration at the terminal server; the terminal server manager can move and change devices without causing errors in applications. However, to ensure that the application connects to both a particular port and service, you can specify both a port and service name.

For more information on setting up terminal servers, refer to the *Local Area Transport (LAT) Network Concepts* manual and the management guide for your terminal server.

6.2 LAT Application Programming

This section describes the basic steps in writing a LAT application:

1. Coordinating available resources
2. Attaching the terminal
3. Setting the terminal characteristics
4. Establishing the connection
5. Performing read and write operations
6. Terminating the connection

6.2.1 Coordinating Available Resources

Before writing your application, you need information about the resources available at your site. If your site includes a terminal server manager and network manager, coordinating with them is important. They can ensure that your application gets the resources it needs and conforms to the existing LAT environment.

First, you need information about the terminal server your application will access. The following list points out some questions to which you need answers. If your site has a terminal server manager, ask that person the questions in the first column; if not, enter the commands in the second column.

Ask the server manager:	**Or enter this command:**
What is the terminal server name?	SHOW SERVER
What are the port names?	SHOW PORT ALL
What service names have been assigned?	SHOW SERVICES/LOCAL This command gives a command syntax error if the terminal does not support service names.

Next, find out what LAT terminals on the host are available for your application to use. If a system manager or network manager is available, ask what terminals you can use. If not, answer the questions in the first column of the following list by entering the commands in the second column of the list.

Ask the network manager:	Or enter this command:
What LAT terminals exist on the host?	MCR command: CON DISPLAY ATTRIBUTES FOR *LH
Which of the LAT terminals are in use by or reserved for other applications?	LCP command: SHOW PORT/APPLICATION

You and the network manager must decide whether to specify the application's target port and/or service from LCP, or through the application.

Coordination with the network manager is important because LCP commands affect applications. For example:

- CREATE and START are required, before you run the application, to create the terminals and start the LAT process. Your network manager may also choose to reserve some terminals for applications at creation time.
- SET PORT lets the network manager specify the target for a LAT terminal, making that terminal an application terminal. Using LCP in this way readies the terminal for you and lets you omit a step in the application.
- DISCONNECT can terminate a connection.
- STOP halts the LAT process, aborting all LAT connections and destroying the target information for application terminals.

For more information on LCP commands, refer to the *DECnet–RSX Guide to Network Management Utilities*.

6.2.2 Attaching the Terminal

Attaching the terminal ensures that your application gets exclusive use of the terminal and prevents it from receiving I/O requests from other applications. It is an optional step, but is very important and strongly recommended. If you omit this step, another application can attach the terminal, specify target information that overrides the information about your target, or perform other disruptive operations.

Attach the terminal by issuing an IO.ATT directive.

6.2.3 Setting the LAT Terminal Characteristics

Next, specify the target terminal server port and/or service to which the terminal will later connect. The terminal server name, port name, and service name are the LAT terminal characteristics. Issue a Set Multiple Characteristics (SF.SMC) QIO, and specify these names by including the TC.MAP characteristics block. Section 6.3.4.1 describes the TC.MAP characteristics block.

Setting these characteristics sets the terminal nobroadcast (NOBRO) and slave characteristics. NOBRO prevents the terminal from receiving broadcasts; SLAVE prevents it from accepting unsolicited input.

If the terminal is currently an interactive terminal, setting the characteristics to provide the target information changes it into an application terminal. If the terminal is currently an application terminal for which LCP previously specified a different target, your characteristics directive resets the characteristics to your target specifications. If the terminal is currently a reserved application terminal, it has no prior target information, but requires this information before you issue a connect request.

You can set the characteristics only if the terminal is currently available for a session. If the terminal is engaged in a session on behalf of an interactive user or another application, the characteristics directive returns an error.

You can omit setting the LAT characteristics in your application if your network manager has already used the LCP SET PORT command to do so. Using LCP in this way has some benefits; for example, it lets you designate a terminal for use by a specific application, whether or not the application is currently in use. Using LCP also allows you to use an existing application designed for a hard-wired device with a remote LAT device without modifying the application, as the next section explains.

6.2.4 Establishing the Connection

The application terminal establishes a network connection in order to exchange data with a remote LAT device. The LAT protocol includes a master/slave relationship: terminal servers can establish sessions with hosts, but hosts cannot establish sessions with terminal servers. When your application initiates a connection, therefore, it actually sends a request soliciting the connection from the terminal server. You can request the connection explicitly or implicitly.

An explicit connection starts when an application issues an Originate Explicit Connection (IO.ORG) directive. This directive solicits a connection between the application terminal and the terminal server specified in the Set Characteristics or LCP SET PORT operation. The IO.ORG directive is for LAT terminals only and is described in Section 6.3.2.

An implicit connection starts when an application immediately issues a Read Virtual Block (IO.RVB) or Write Virtual Block (IO.WVB) directive, instead of first requesting a connection. If the terminal is a LAT application terminal that is not yet connected to a remote terminal server, the initial read or write request starts a connection sequence. When the connection completes, the read or write operation is performed.

Implicit connections let you run an application originally written for a hardwired device in a LAT environment. In most cases, you need not modify the application if you use LCP to specify the target for the application terminal. Note, however, that the initial IO.RVB or IO.WVB directive will return status messages that report the success or failure of the connection attempt. Refer to Section 6.3.3 for the LAT connection status returns.

Some target services, such as printers, have queues. If you make a connection request to a queued service, the connection does not complete until the service becomes available. To avoid an indefinite wait, include a time-out routine in your application.

6.2.5 Reading and Writing Data

The Read Virtual Block (IO.RVB) and Write Virtual Block (IO.WVB) QIO functions exchange data with the application terminal. Use them as you do for standard terminals, but remember that if you have not created an explicit connection, the first read or write request establishes an implicit connection. Use time-out and error-handling routines with the first read or write request in case connection problems occur.

6.2.6 Terminating the Connection

Termination methods for explicit and implicit connections differ.

An explicit connection terminates only in response to a Disconnect Terminal (IO.HNG) directive. IO.HNG assures a clean, synchronous termination and does not complete until the disconnect sequence completes. Once the directive completes, the remote device becomes available for I/O from other sources. By not issuing an IO.HNG directive, an application that exits and restarts can maintain an available connection.

Implicit connections terminate in response to both the IO.HNG and Detach I/O Device (IO.DET) directives and at task exit. Both IO.HNG and IO.DET assure clean, synchronous terminations and, on completion, leave the remote device available for I/O from other sources. Use IO.DET, however, only if you previously attached the terminal.

Although task exit can terminate an implicit connection, avoid using it as a termination method. If an application includes multiple tasks, the connection terminates when any one task exits, and the next read or write request establishes a new connection. Terminating and reconnecting consumes time and adds overhead. It can also cause you to lose the remote resource if another application is queued and waiting when yours disconnects. To avoid this, do one of the following:

- Modify the application to issue an IO.ORG directive.
- Create an initial task that issues an IO.ORG directive and then passes control to the existing application.

Both IO.HNG and IO.DET allow the application terminal to send outstanding data to the terminal server before completing a disconnect request.

Also note that any connection can terminate due to unexpected errors such as terminal server crash or network errors, or because the network manager has issued an LCP STOP or LCP TERMINATE command.

6.2.7 Summary

Table 6–1 summarizes the basic steps in a LAT application. The first column lists the functions to perform. The second and third columns show what you do to perform that function using explicit and implicit connections, respectively.

Table 6–1: Steps in a LAT Application

Function	Explicit Connection	Implicit Connection
Attach the terminal.	Issue IO.ATT.	Same.
Set LAT terminal characteristics.	Issue SF.SMC and include the TC.MAP characteristic. Alternatively, use the LCP SET PORT command prior to execution.	Same.
Establish a connection.	Issue IO.ORG.	Issue an initial IO.RVB or IO.WVB directive.
Application executes....		
Terminate the connection.	Issue IO.HNG.	Issue IO.HNG or IO.DET. (Use IO.DET only if you previously attached the terminal.) Task exit also terminates the connection.

6.3 Directives for Programming Application Terminals

The LAT application terminals use standard RSX terminal driver directives. Table 6–2 notes the special usage of directives that form important steps in a LAT application and gives the formats of those directives.

Table 6–2: Terminal Driver Directive Usage for LAT Terminals

Directive	Use and Format
IO.ATT	Attaches the terminal. QIOW$C IO.ATT,*lun*,*efn*,[*pri*],[*isb*],[*ast*]
IO.DET	Terminates an implicit connection to a LAT terminal that you previously attached. QIOW$C IO.DET,*lun*,*efn*,[*pri*],[*isb*],[*ast*]
IO.HNG	Terminates any connection to a LAT terminal. QIOW$C IO.HNG,*lun*,*efn*,[*pri*],[*isb*],[*ast*]
IO.ORG	Initiates an explicit connection between a LAT application terminal and the terminal server previously specified in an SF.SMC directive or LCP SET PORT command. QIOW$C IO.ORG,*lun*,*efn*,[*pri*],[*isb*],[*ast*]
IO.RVB	Reads data from a LAT application terminal. If the application terminal is not yet connected to the remote terminal server, this directive first creates the connection and then performs the read operation. QIOW$C IO.RVB,*lun*,*efn*,[*pri*],[*isb*],[*ast*],<*stadd*,*size*,*pn*>
IO.WVB	Writes data to a LAT application terminal. If the application terminal is not yet connected to the remote terminal server, this directive first creates the connection and then performs the write operation. QIOW$C IO.WVB,*lun*,*efn*,[*pri*],[*isb*],[*ast*],<*stadd*,*size*,*pn*>
SF.GMC	Returns the name of the terminal server and port and/or service to which a LAT application terminal is mapped. Also returns the terminal's connection status. QIOW$C SF.GMC,*lun*,*efn*,[*pri*],[*isb*],[*ast*],<*stadd*,*size*,*pn*>
SF.SMC	Maps the LAT application terminal to a specified terminal server port and/or service to which it can later connect. QIOW$C SF.SMC,*lun*,*efn*,[*pri*],[*isb*],[*ast*],<*stadd*,*size*,*pn*>

The *RSX–11M/M–PLUS I/O Drivers Reference Manual* and *Micro/RSX I/O Drivers Reference Manual* have information on most of the directives you use. However, the following information applies only to LAT terminals:

- The IO.ORG directive.
- Status codes that result from connection solicitation requests.
- LAT-specific characteristics to use with Set Multiple Characteristics and Get Multiple Characteristics directives.

6.3.1 Programming Suggestions

The following suggestions may be helpful in writing your application:

- Use the Wait form of the QIO or an AST routine to synchronize the application with the completion of the connection. Using the Wait QIO form ensures that connection completes before the application proceeds.
- Check the status block after every read and write request. Because your connection to a LAT device is through the network, and not direct, check the status block to make sure that the write request succeeded.
- Remember in using all RSX directives that LAT devices are remote, not hardwired.

Occasionally, you might get a Device Not Ready (IE.DNR) error when attempting to perform a Read, Write, or Set Characteristics operation on a terminal that you have already attached successfully. This error can occur when a previous application has issued an IO.DET directive to terminate its connection. Upon receiving this directive, a terminal can detach and be ready for a new attachment even if the previous application's connection is completely terminated or its IO.DET directive completed. In this case, your application can attach a terminal whose previous session is not completely finished. To prevent your own application from causing this type of problem, issue an IO.HNG directive before each IO.DET. The IO.HNG cleanly breaks the connection before the IO.DET detaches the terminal.

IO.ORG
(Originate Explicit Connection)

6.3.2 IO.ORG — Originate Explicit Connection

IO.ORG solicits a terminal server to initiate a connection to the LAT terminal. The directive does not complete until the connection is established or an error occurs. Ensure that the application waits for the connection by using the Wait QIO form or an AST routine. The LAT terminal characteristics must be set before you issue a connection request.

IO.ORG is a control function; it does not cause any data transfer.

Format:

QIOW$C IO.ORG,*lun*,*efn*,[*pri*],[*isb*],[*ast*]

Parameters:

The following are the IO.ORG parameters. For more information on QIO parameters, refer to the *RSX-11M-PLUS* or *Micro/RSX I/O Drivers Reference Manual*.

Parameter	Meaning
lun	The logical unit number associated with the physical device that the I/O request accesses.
efn	The number of the event flag to associate with the QIO operation. An event flag is required with the Wait macro form.
pri	A priority value for compatibility with IAS. This parameter must have a value of zero (0) or null.
isb	The address of the I/O status block, a 2-word array that contains the completion status for the I/O request on completion of the operation (see under "Status Returns").
ast	The address of an optional user-written routine to execute after this call completes. When control branches to the specified address, it has the software priority of the requesting task. For no AST processing, omit the parameter or enter the value zero (0).

Status Returns:

The I/O status block that *isb* specifies has the following format:

LKG-1261-87

The completion codes are described in the next section.

6.3.3 Status Codes for LAT Connections

The following status codes result from the connection solicitation sequence. The sequence can be a result of an explicit (IO.ORG) or implicit (IO.RVB or IO.WVB) connection request.

Symbol Name	Decimal Value	Octal Value	Meaning
IS.SUC	1	1	Success.
IE.RSU	-17	347	Shared resource in use. The terminal is connected or busy with a connect request.
IE.DNR	-3	375	Device not ready.

A status block that returns IE.DNR in the low byte returns one of the following in the high byte:

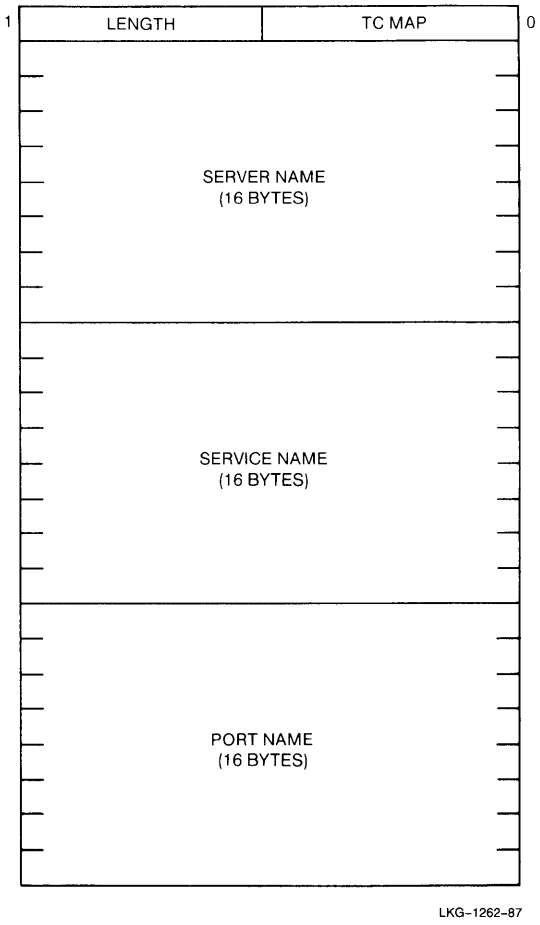
Symbol Name	Decimal Value	Octal Value	Meaning
	0	0	The terminal server or application terminal was unavailable or incorrectly specified.
IE.PRI	-16	360	The terminal server's group access list does not include your host.
IE.ICE	-47	321	Internal corruption error.
IE.NRJ	-74	266	Service busy.
IE.FLN	-81	251	Terminal server service disabled.
IE.CNR	-96	240	Terminal not a LAT application terminal.
IE.UKN	-97	237	The specified terminal server port or service does not exist.
IE.IRR	-102	232	Insufficient resources at the terminal server.

6.3.4 LAT Specific Characteristics for SF.GMC

The Get Multiple Characteristics (SF.GMC) directive returns information about various characteristics. This section describes only the characteristics specific to LAT application terminals: TC.MAP and TC.QDP. Using TC.MAP returns the application terminal's associated terminal server, remote port, and/or service name. Using TC.QDP returns the application terminal's connection status and queue position. Use this information in conjunction with the information on SF.GMC in the *RSX-11M/M-PLUS* or *Micro/RSX I/O Drivers Reference Manual*.

6.3.4.1 TC.MAP

With the TC.MAP characteristic, the SF.GMC directive returns the name of a terminal server. It also returns either a service name or port name, or both. The block has the following format.



where

Length — is the length of the characteristics block. The length value 48. indicates the presence of characteristics data; 0 indicates no data present.

Server name — is the name of a terminal server, in up to 16. bytes, padded with zeroes.

Service name — is a service name, in up to 16. bytes, padded with zeroes. Either the service or port name must be present; both can be present. If no service name is returned, the first byte contains 0.

Port name — is a port name, in up to 16. bytes, padded with zeroes. Either the service or port name must be present; both can be present. If no port name is returned, the first byte contains 0.

The following programming fragment shows the format for setting up the TC.MAP characteristic to use with SF.GMC:

```
CHRBUE:   .BYTE    IC.MAP       ; Characteristic value
          .BYTE    48.          ; Length of data buffer
;
SERVER:   .BLKB    16.          ; Returns the terminal server name.
;
SERVICE:  .BLKB    16.          ; Returns the service name
                                ; or zero (0) for no service name.
;
PORT:     .BLKB    16.          ; Returns the port name or
                                ; zero (0) for no port name.
;
CHRLEN=.-CHRBUE
```

6.3.4.2 TC.QDP

With the TC.QDP characteristic, SF.GMC returns the status of the connection request. If the connection request is pending at a queued service, SF.GMC also returns the queue position. The block has the following format.

LKG-1263-87

where

Status	returns one of the following values:
0	Not connected
1	Connected
< 1	Pending. A value greater than 1 indicates the request's position in the queue.

6.3.5 LAT Specific Characteristics for SF.SMC

The Set Multiple Characteristics (SF.SMC) directive enables a task to set and reset the characteristics of a terminal. This section describes only the LAT characteristics. For information on other characteristics, refer to the *RSX–11M/M–PLUS* or *Micro/RSX I/O Drivers Reference Manual*.

Using the TC.MAP characteristic, this directive supplies the information on the target terminal server port and/or service for the application terminal. Specifying the target of its future connection defines a terminal as an application terminal.

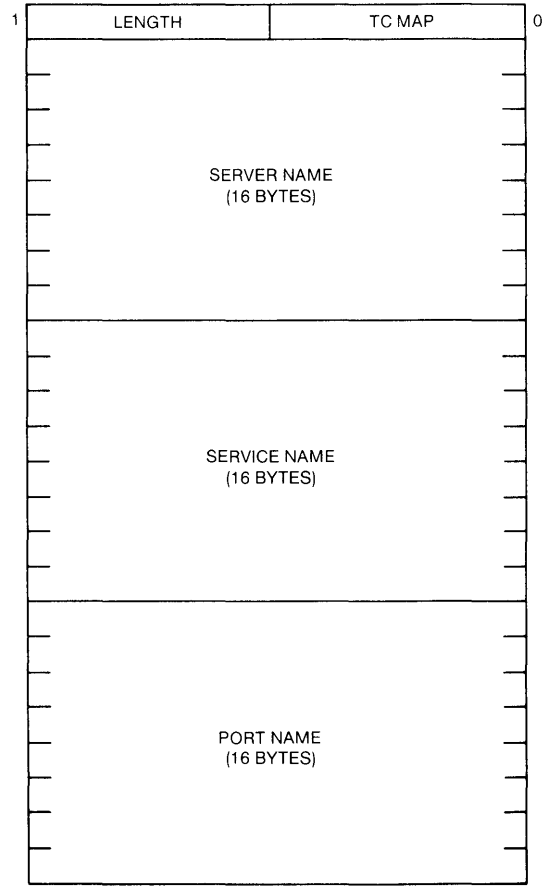
Each SF.SMC directive resets the characteristics. The characteristics you specify supersede any previous settings, including those specified through LCP.

6.3.5.1 TC.MAP

The TC.MAP characteristic specifies or clears the terminal server name and the port and/or service name.

To set the LAT characteristics, enter 48. in the length field and include the terminal server name. Also specify the service name or port name, or both. To clear the LAT characteristics, enter 0 in the length field. Clearing these characteristics returns an application terminal to its previous state as a reserved application terminal or an interactive terminal.

The block has the following format.



where

Length	is the characteristics block length. Use 48. to specify characteristics; use 0 to clear characteristics.
Server name	is the name of the terminal server, in 16. or fewer bytes. End the name in a zero byte if it is smaller than 16. bytes.
Service name	is the name of a service that the terminal server offers. To omit the service name, allocate 16. bytes and enter 0 in the first and last bytes of the field. End the name in a zero byte if it is smaller than 16. bytes.
Port name	is the name of a terminal server port. To omit the port name, allocate the full 16. bytes and enter 0 in the first and last bytes of the field. End the name in a zero byte if it is smaller than 16. bytes.

Using this directive with TC.MAP to set LAT characteristics:

- Sets the specified terminal to SLAVE and NOBROADCAST.
- Sets the S6.LAT bit in unit status word 6 (U.TST6) in the Unit Control Block for the terminal. This bit indicates that the terminal is an application terminal.

Using this directive with TC.MAP to clear characteristics returns the terminal to its previous state as a reserved application terminal or an interactive terminal available to interactive users or applications. If the terminal is connected, when you issue the directive, however, an IE.RSU error results.

Status Returns:

In addition to the standard SF.SMC status returns, IE.RSU applies only to LAT application terminals.

Symbol Name	Decimal Value	Octal Value	Meaning
IE.RSU	-17	357	Shared resource in use. The terminal is busy with another session or connection request.

The following fragment shows the format for setting up the TC.MAP characteristic:

```
CHRBUF:  .BYTE     TC.MAP            ; Characteristic value
         .BYTE     48.               ; Length of data buffer
;
SERVER:  .ASCIZ    /Server_name/     ; Server name to 16. bytes
A=.-SERVER                           ; Length of service name
         .BLKB     <16.-A>           ; Pad with zeroes to 16. bytes)
;
SERVICE: .ASCIZ    /Service_name/    ; Service name to 16. bytes
                                     ; or zero (0) to omit name
B=.-SRVICE                           ; Length of service name
         .BLKB     <16.-B>           ; Pad with zeroes to 16. bytes)
;
PORT:    .ASCIZ    /Port_name/       ; Port name to 16. bytes
                                     ; or zero (0) to omit name
C=.-PORT                             ; Length of port name
         .BLKB     <16.-C>           ; Pad with zeroes to 16. bytes.
;
CHRLEN=.-CHRBUF
```

6.4 LAT Application Programming Examples

The following examples show how you can write to and read from a device attached to a remote terminal server. The first example, LATORG, uses the explicit connection method. The second example, LATEX, uses the implicit method.

6.4.1 Explicit Connection Example

```
        .TITLE  LATORG
        .IDENT  /Y1.0/
;
;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
;
; This program demonstrates the use of LAT application terminals,
; using the explicit method for establishing the connection
; between the remote device and host terminal.
;
; First the program specifies the application terminal's target.
; Next, it issues an IO.ORG QIO to initiate an explicit connection.
;
; If the connection is successful, the program displays a message on the
; remote device and prompts the user for input.  CTRL/Z terminates the
; session.
;
; Data entered at the remote device is printed on the local TI:.
;
; The example uses TT30: as the host terminal.  It writes data to a port
; named PORT_2 on Terminal Server AUDIO.
;
; Assembly command:  MAC LATORG,LATORG=LATORG
;
; To task build, enter the following command at the TKB> prompt:
;
; LATORG/PR:0,LATORG=LATORG
; /
; TASK=...LAT
; //
;
;
        .MCALL  QIOW$,QIOW$C,ALUN$C,DIR$,EXIT$S
;
START:  ALUN$C  1,TT,30                 ; Assign LUN 1 to TT30:
        QIOW$C  IO.ATT,1,1,,IOSB        ; Attach the terminal
```

(continued on next page)

```
;
        DIR$    #SMC                    ; Specify the terminal's target
        BMI     5$                      ; If MI, directive error
        TSTB    IOSB                    ; Check status
        BPL     10$                     ; If PL, continue
5$:     MOV     #SMCERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        JMP     EXIT                    ; Exit
;
; Now, initiate the connection.
;
10$:    QIOW$C  IO.ORG,1,1,,IOSB        ; Initiate the connection
        BMI     20$                     ; If MI, directive error
        TSTB    IOSB                    ; Check status
        BPL     30$                     ; If PL, connection OK - branch
20$:    MOV     #QIOERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        JMP     EXIT                    ; Exit
30$:    MOV     #SUCMSG,R0              ; Get success message text
        CALL    PRINT                   ; Print message
;
; Now the program reads data from the application terminal.  The
; data is printed (displayed) on the local TI:.  The program continues
; to read data from the terminal until the user at the remote device
; enters the terminating control sequence, CTRL/Z.
;
35$:    DIR$    #READ                   ; Read data from application terminal
        BMI     40$                     ; If MI, directive error
        TSTB    IOSB                    ; Check status
        BPL     50$                     ; If PL, success - branch
        CMPB    IOSB,#IE.EOF            ; Did user type CTRL/Z?
        BEQ     60$                     ; If EQ, yes - not an error
40$:    MOV     #REAERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        BR      EXIT                    ; And exit
;
50$:    MOV     #INHEAD,WRITE5+Q.IOPL   ; Move output buffer address to DPB
        MOV     IOSB+2,WRITE5+Q.IOPL+2  ; Move the length of received data
        ADD     #8.,WRITE5+Q.IOPL+2     ; Add length of header
        DIR$    #WRITE5                 ; Print the data
        BR      35$                     ; Loop to read more data
60$:
        MOV     #BYEMSG,WRITE5+Q.IOPL   ; Move end message to DPB
        MOV     #BYEMSL,WRITE5+Q.IOPL+2 ; Move length of message
        DIR$    #WRITE5                 ; Print terminating message
;
EXIT:   QIOW$C  IO.HNG,1,1              ; Disconnect the session
        QIOW$C  IO.DET,1,1              ; Detach the terminal
        EXIT$S                          ; And exit
;
;
; **-ERROR - Error handling routine.  This routine formats an error
; message and prints it on the local TI:
;
; INPUTS: R0 - Pointer to the error message text (must end in a 0 byte)

;         $DSW - Directive status from last directive executed
;         IOSB - I/O status if $DSW is positive (no error)
;
;       Registers altered: R0
;
;
```

```
ERROR:  MOV     R1,-(SP)                ; Save R1
        CALL    PRINT                   ; Print message
        CLR     R1                      ; Clear storage for error status
        TSTB    $DSW                    ; Check for directive error
        BPL     30$                     ; If PL, no directive error
        MOV     #DSWERR,R0              ; Point at error text (DSW)
        CALL    PRINT                   ; Print error message
        BISB    $DSW,R1                 ; Get error status to R1
        CALL    CONVRT                  ; Convert and print error message
        BR      EX                      ; Branch to common exit
;
30$:    MOV     #IOSBM1,R0              ; Point at message text
        CALL    PRINT                   ; Print IOSB message (first half)
        BISB    IOSB,R1                 ; Get low byte of IOSB
        CALL    CONVRT                  ; Convert and print low byte of IOSB
        MOV     #IOSBM2,R0              ; Point at 2nd half of IOSB message text
        CALL    PRINT                   ; Print the message
        CLR     R1                      ; Clear storage
        BISB    IOSB+1,R1               ; Get value of 2nd half of IOSB
        CALL    CONVRT                  ; Convert and print 2nd half of IOSB
;
EX:     MOV     (SP)+,R1                ; Restore register
        RETURN                          ; Back to caller
;
; **-CONVRT - Convert octal to ASCII and print it out
;
; INPUTS:  R1 - Byte value to be converted
;
;     All registers are preserved
;
;
CONVRT: MOV     R1,-(SP)                ; Save R1
        MOV     R2,-(SP)                ; Save R2
        MOV     R3,-(SP)                ; Save R3
        CLR     R3                      ; Clear storage
        MOV     #$TEMP+4,R2             ; Point at temporary storage area
        CLRB    -(R2)                   ; Ensure 0 byte at the end
10$:    MOV     R1,R3                   ; Get value to R3
        BIC     #177770,R3              ; Clear all but the last 3 bits
        ADD     #60,R3                  ; Convert to ASCII
        MOVB    R3,-(R2)                ; Save ASCII in buffer
        CMP     R2,#$TEMP               ; Are we done?
        BEQ     20$                     ; If EQ, no
        ASR     R1                      ; Shift over 3 bits
        ASR     R1                      ; ***
        ASR     R1                      ; ***
        BR      10$                     ; Loop for more
;
20$:    MOV     #$TEMP,R0               ; Point at ASCII
        CALL    PRINT                   ; Print it out
        MOV     (SP)+,R3                ; Restore registers
        MOV     (SP)+,R2                ;  ***
        MOV     (SP)+,R1                ;  ***
        RETURN                          ; Back to caller
;
;
; **-PRINT - Print a message on the local TI:
;
; INPUTS:  R0 -  Address of string to print
;
;       Note:  The ASCII string must end in a 0 byte
;
;       All registers are preserved
;
;
```

(continued on next page)

```
PRINT:  MOV     R0,-(SP)                ; Save pointer
        CLR     -(SP)                   ; Clear storage
10$:    TSTB    (R0)+                   ; Look for end
        BEQ.    20$                     ; If EQ, done
        INC     (SP)                    ; Count a character
        BR      10$                     ; Check next
20$:    MOV     (SP)+,QIO+Q.IOPL+2      ; Move length to DPB
        MOV     (SP),QIO+Q.IOPL         ; Move address to DPB
        DIR$    #QIO                    ; Print the message
        MOV     (SP)+,R0                ; Restore pointer to message
        RETURN                          ; Back to caller
;
QIO:    QIOW$   IO.WVB,5,1,,,,<0,0>
;
WRITE:  QIOW$   IO.WVB,1,1,,IOSB,,<OUTBUF,OUTLEN>
;
WRITE5: QIOW$   IO.WVB,5,1,,,,<0,0>
;
READ:   QIOW$   IO.RPR,1,1,,IOSB,,<INBUF,INLEN,,PROMPT,PROLEN>
;
SMC:    QIOW$   SF.SMC,1,1,,IOSB,,<CHRBUF,CHRLEN> ;DPB for SMC
;
CHRBUF: .BYTE   TC.MAP                  ; Characteristic name
        .BYTE   48.                     ; Characteristics buffer length
;
SERVER: .ASCIZ  /AUDIO/                 ; Server name (ends in 0 byte)
A=.-SERVER                              ; Actual server name length
        .BLKB   <16.-A>                 ; Allocate the full 16. bytes for name
;
SERVIC: .BYTE   0                       ; Ensure 0 byte (no service name)
        .BLKB   15.                     ; Fill to 16. bytes
;
PORT:   .ASCIZ  /PORT_2/                ; Port name
A=.-PORT                                ; Length of actual port name
        .BLKB   <16.-A>                 ; Allocate the full 16. bytes for port
;
CHRLEN=.-CHRBUF                         ; Length of entire buffer
;
;
IOSB:   .WORD   0,0                     ; IOSB for write and read operations
$TEMP:  .BLKB   4                       ; Temporary storage for error routine
INHEAD: .ASCII  <12><15>/DATA> /        ; Header for printing data to local TI:
INBUF:  .BLKB   132.                    ; Input buffer
INLEN=.-INBUF
;
PROMPT: .ASCII  <12><15>/Enter data>/
PROLEN=.-PROMPT
        .EVEN
OUTBUF: .ASCII  <12><15><7>/Application terminal now active/
OUTLEN=.-OUTBUF
DSWERR: .ASCIZ  <12><15>/Directive Error code ($DSW) = /
IOSBM1: .ASCIZ  <12><15>/IOSB return code - Low byte = /
IOSBM2: .ASCIZ  /   High byte = /
;
SMCERR: .ASCIZ  <12><15>/Error setting terminal characteristics./
QIOERR: .ASCIZ  <12><15>/Error establishing implicit connection./
SUCMSG: .ASCIZ  <12><15>/Connection established to terminal server./
REAERR: .ASCIZ  <12><15>/Error reading data from application terminal./
BYEMSG: .ASCII  <12><15>/Read terminated by user. Now disconnecting/
        .ASCIZ  / application terminal./
BYEMSL=.-BYEMSG
        .EVEN
        .END    START
```

6.4.2 Implicit Connection Example

```
        .TITLE LATEX
        .IDENT  /Y1.0/
;
;
; Copyright (C) 1983, 1985, 1986, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
;
; This software is furnished under a license and may be used and  copied
; only  in  accordance  with  the  terms  of  such  license and with the
; inclusion of the above copyright notice.  This software or  any  other
; copies  thereof may not be provided or otherwise made available to any
; other person.  No title to and ownership of  the  software  is  hereby
; transferred.
;
; The information in this software is subject to change  without  notice
; and  should  not  be  construed  as  a commitment by Digital Equipment
; Corporation.
;
; Digital assumes no responsibility for the use or  reliability  of  its
; software on equipment which is not supplied by Digital.
;
;
; This program demonstrates the use of LAT application terminals,
; using the implicit method for establishing the connection
; between the host terminal and remote device.
;
; First the program specifies the target of the application terminal.
; Next, it posts a write request to the terminal to establish the
; connection.  If the connection completes successfully, the program displays
; a message on the remote device and prompts the user for input.
; Any data that the user enters at the remote device is printed (displayed)
; on the local TI:.  The user enters CTRL/Z to terminate the session.
;
; The example uses TT30: as the application terminal.  It writes data to
; the port PORT_2 on terminal server AUDIO.
;
; Assembly command:  MAC LATEX,LATEX=LATEX
;
; To task build, enter the following command at the TKB> prompt:
;
; LATEX/PR:0,LATEX=LATEX
; /
; TASK=...LAT
; //
;
;
        .MCALL  QIOW$,QIOW$C,ALUN$C,DIR$,EXIT$S
;
START:  ALUN$C  1,TT,30                 ; Assign LUN 1 to TT30:
        QIOW$C  IO.ATT,1,1,,IOSB        ; Attach the terminal
;
        DIR$    #SMC                    ; Specify its target
        BMI     5$                      ; If MI, directive error
```

(continued on next page)

```
        TSTB    IOSB                    ; Check status
        BPL     10$                     ; If PL, continue
5$:     MOV     #SMCERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        JMP     EXIT                    ; And exit


; Now, write data to the LAT application terminal and initiate
; the implicit connection sequence.
;
10$:    DIR$    #WRITE                  ; Initiate connection and write data
        BMI     20$                     ; If MI, directive error
        TSTB    IOSB                    ; Check completion status
        BPL     30$                     ; If PL, success - branch
20$:    MOV     #QIOERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        JMP     EXIT                    ; And exit..
30$:    MOV     #SUCMSG,R0              ; Get success message text
        CALL    PRINT                   ; Print message
;           .
; Now the program reads data from the remote device.  The  data is
; printed on the local TI:.  The program continues to read
; data until the user at the remote device enters the terminating
; control character sequence, CTRL/Z.
;
35$:    DIR$    #READ                   ; Read data from remote device
        BMI     40$                     ; If MI, directive error
        TSTB    IOSB                    ; Check status
        BPL     50$                     ; If PL, success - branch
        CMPB    IOSB,#IE.EOF            ; Did user type CTRL/Z?
        BEQ     60$                     ; If EQ, yes - not an error
40$:    MOV     #REAERR,R0              ; Get error message text
        CALL    ERROR                   ; Call error routine
        BR      EXIT                    ; And exit
;
50$:    MOV     #INHEAD,WRITE5+Q.IOPL   ; Move output buffer address to DPB
        MOV     IOSB+2,WRITE5+Q.IOPL+2  ; Move the length of received data
        ADD     #8.,WRITE5+Q.IOPL+2     ; Add length of header
        DIR$    #WRITE5                 ; Print the data
        BR      35$                     ; Loop to read more data
60$:
        MOV     #BYEMSG,WRITE5+Q.IOPL   ; Move end message to DPB
        MOV     #BYEMSL,WRITE5+Q.IOPL+2 ; Move length of message
        DIR$    #WRITE5                 ; Print terminating message
;
; Now the program detaches the terminal. If the connection sequence
; was successful, the detach request terminates the connection.
; This is the recommended termination method for implicit connections.
;
EXIT:   QIOW$C  IO.DET,1,1              ; Detach the terminal, disconnect
        EXIT$S                          ; And exit
;
;
        .SBTTL ERROR - Error handling routines
;
```

```
; **-ERROR - Error handling routine.  This routine formats an error
; message and prints it on the local TI:
;
; INPUTS: R0 - Pointer to the error message text (must end in a 0 byte)
;         $DSW - Directive status from last directive executed
;         IOSB - I/O status if $DSW is positive (no error)
;
;       Registers altered: R0
;
;
ERROR:  MOV     R1,-(SP)                ; Save R1
        CALL    PRINT                   ; Print message
        CLR     R1                      ; Clear storage for error status
        TSTB    $DSW                    ; Check for directive error
        BPL     30$                     ; If PL, no directive error
        MOV     #DSWERR,R0              ; Point at error text (DSW)
        CALL    PRINT                   ; Print error message
        BISB    $DSW,R1                 ; Get error status to R1
        CALL    CONVRT                  ; Convert and print error message
        BR .    EX                      ; Branch to common exit
;
30$:    MOV     #IOSBM1,R0              ; Point at message text
        CALL    PRINT                   ; Print IOSB message (first half)
        BISB    IOSB,R1                 ; Get low byte of IOSB
        CALL    CONVRT                  ; Convert and print low byte of IOSB
        MOV     #IOSBM2,R0              ; Point at 2nd half of IOSB message text
        CALL    PRINT                   ; Print the message
        CLR     R1                      ; Clear storage
        BISB    IOSB+1,R1               ; Get value of 2nd half of IOSB
        CALL    CONVRT                  ; Convert and print 2nd half of IOSB
;
EX:     MOV     (SP)+,R1                ; Restore register
        RETURN                          ; Back to caller
;
; **-CONVRT - Convert octal to ASCII and print it out
;
; INPUTS:  R1 - Byte value to convert
;
;     All registers are preserved
;
;
CONVRT: MOV     R1,-(SP)                ; Save R1
        MOV     R2,-(SP)                ; Save R2
        MOV     R3,-(SP)                ; Save R3
        CLR     R3                      ; Clear storage
        MOV     #$TEMP+4,R2             ; Point at temporary storage area
        CLRB    -(R2)                   ; Ensure 0 byte at the end
10$:    MOV     R1,R3                   ; Get value to R3
        BIC     #177770,R3              ; Clear all but the last 3 bits
        ADD     #60,R3                  ; Convert to ASCII
        MOVB    R3,-(R2)                ; Save ASCII in buffer
        CMP     R2,#$TEMP               ; Are we done?
        BEQ     20$                     ; If EQ, no
        ASR     R1                      ; Shift over 3 bits
        ASR     R1                      ; ***
```

(continued on next page)

```
        ASR     R1                      ; ***
        BR      10$                     ; Loop for more
;
20$:    MOV     #$TEMP,R0               ; Point at ASCII
        CALL    PRINT                   ; Print it
        MOV     (SP)+,R3                ; Restore registers
        MOV     (SP)+,R2                ;  ***
        MOV     (SP)+,R1                ;  ***
        RETURN                          ; Back to caller
;
;
; **-PRINT - Prints a message on the local TI:
;
; INPUTS:  R0 -  Address of string to print
;
;       Note:  The ASCII string must end in a 0 byte
;
;       All registers are preserved
;
;
PRINT:  MOV     R0,-(SP)                ; Save pointer
        CLR     -(SP)                   ; Clear storage
10$:    TSTB    (R0)+                   ; Look for end
        BEQ     20$                     ; If EQ, done
        INC     (SP)                    ; Count a character
        BR      10$                     ; Check next
20$:    MOV     (SP)+,QIO+Q.IOPL+2      ; Move length to DPB
        MOV     (SP),QIO+Q.IOPL         ; Move address to DPB
        DIR$    #QIO                    ; Print the message
        MOV     (SP)+,R0                ; Restore pointer to message
        RETURN                          ; Back to caller
;
;
        .SBTTL  DATA - Data areas, messages and DPBs
;
; **-DATA - Data areas, messages and DPBs
;
; Directive Parameter blocks (DPBs)
;
QIO:    QIOW$   IO.WVB,5,1,,,,<0,0>
;
WRITE:  QIOW$   IO.WVB,1,1,,IOSB,,<OUTBUF,OUTLEN>
;
WRITE5: QIOW$   IO.WVB,5,1,,,,<0,0>
;
READ:   QIOW$   IO.RPR,1,1,,IOSB,,<INBUF,INLEN,,PROMPT,PROLEN>
;
;
; Set characteristics DPB and characteristics buffer
;
SMC:    QIOW$   SF.SMC,1,1,,IOSB,,<CHRBUF,CHRLEN> ;DPB for SMC
;
CHRBUF: .BYTE   TC.MAP                  ; Characteristic name
        .BYTE   48.                     ; Characteristics buffer length
;
```

```
SERVER: .ASCIZ  /AUDIO/                 ; Server name (ends in 0 byte)
A=.-SERVER                              ; Actual server name length
        .BLKB   <16.-A>                 ; Allocate the full 16. bytes for name
;
SERVIC: .BYTE   0                       ; Ensure 0 byte (no service name)
        .BLKB   15.                     ; Fill to 16. bytes
;
PORT:   .ASCIZ  /PORT_2/                ; Port name
A=.-PORT                                ; Actual port name length
        .BLKB   <16.-A>                 ; Allocate the full 16. bytes
;
CHRLEN=.-CHRBUF                         ; Length of entire buffer
;
;
; Data storage and ASCII Text
;
IOSB:   .WORD   0,0                     ; IOSB for write and read operations
$TEMP:  .BLKB   4                       ; Temporary storage for error routine
INHEAD: .ASCII  <12><15>/DATA> /        ; Header for printing data to local TI:
INBUF:  .BLKB   132.                    ; Input buffer
INLEN=.-INBUF
;
PROMPT: .ASCII  <12><15>/Enter data>/
PROLEN=.-PROMPT
        .EVEN
OUTBUF: .ASCII  <12><15><7>/Application terminal now active/
OUTLEN=.-OUTBUF
DSWERR: .ASCIZ  <12><15>/Directive Error code ($DSW) = /
IOSBM1: .ASCIZ  <12><15>/IOSB return code - Low byte = /
IOSBM2: .ASCIZ  /   High byte = /
;
SMCERR: .ASCIZ  <12><15>/Error setting terminal characteristics./
QIOERR: .ASCIZ  <12><15>/Error establishing implicit connection./
SUCMSG: .ASCIZ  <12><15>/Connection established to terminal server./
REAERR: .ASCIZ  <12><15>/Error reading data from application terminal./
BYEMSG: .ASCII  <12><15>/Read terminated by user.  Now disconnecting/
        .ASCIZ  / application terminal./
BYEMSL=.-BYEMSG
        .EVEN
        .END    START
```

A

Disconnect or Reject Reason Codes

The following list contains the error reason codes available at the logical link user interface. These codes can be returned after either of the following events:

- The network rejected a connect request (IE.NRJ).
- The network aborted a connected logical link (NT.ABO).

The symbols in column 1 are defined in the macro NSSYM$. NSSYM$ is located in NETLIB.MLB (moved to LB:[1,1] during network generation). The events in column 5 indicate the condition that occurred. C refers to a connect request and A refers to a network abort.

Symbol Name	Decimal Value	Octal Value	Standard Message/Explanation	Event
NE$RES	1	1	Insufficient network resources The logical link could not be connected because either the local or the remote node had insufficient network resources (for example, insufficient logical links, remote node counters, or dynamic storage region (DSR) on RSX systems).	C
NE$NOD	2	2	Unrecognized node name The logical link could not be connected because the destination node name did not correspond to any known node address.	C

Symbol Name	Decimal Value	Octal Value	Standard Message/Explanation	Event
NE$NSR	3	3	Remote node shutting down The logical link could not be connected because the network on the remote node was in the process of shutting down and would accept no more logical link connections.	C
NE$UOB	4	4	Unrecognized object The logical link could not be connected because the object number or name specified did not exist at the remote node.	C
NE$FMT	5	5	Invalid object name format The logical link could not be connected because the node did not understand the object name format.	C
NE$MLB	6	6	Object too busy The logical link could not be connected because the remote object was too busy handling other logical links.	C
NE$ABM	8	10	Abort by network management The logical link has been aborted by an operator or a program using network management.	A
NE$NNF	10	12	Invalid node name format The logical link could not be connected because the remote node name format was invalid. For example, the name contained illegal characters or was too long.	C
NE$NSL	11	13	Local node shutting down The logical link could not be connected because the network on the local node was in the process of shutting down.	C

Symbol Name	Decimal Value	Octal Value	Standard Message/Explanation	Event
NE$ACC	34	42	Access control rejected The logical link could not be connected because the remote node or object could not understand or would not accept the access control information.	C
NE$ABO	38	46	No response from object The logical link could not be connected because the object did not respond. For example, the object responded too slowly or terminated abnormally.	C
NE$ABO	38	46	Remote node or object failed The connected logical link was aborted because the remote node or the object terminated abnormally.	A
NE$COM	39	47	Node unreachable Either the logical link could not be connected or the connected logical link was aborted because no path existed to the remote node.	C/A

B

Object Types

This appendix lists the object type code values defined by Digital, expressed as octal and decimal byte values. Digital reserves the right to add object types and to make changes to the descriptor formats used by the object types. At present, a descriptor format of 1 indicates a user process (object type 000). All other object types in the list have a descriptor format of 0, requiring definition by the object type codes in the first two columns.

Object Type		Process Type
Octal	Decimal	
000	000	General task, user process
001	001	File Access Listener (FAL/DAP) Version 1
002	002	Unit record services (URDS)
003	003	Application terminal services (ATS)
004	004	Command terminal services (CTS)
005	005	RSX-11M Remote Task Control utility (TCL) Version 1
006	006	Operator services interface
007	007	Node resource manager
010	008	IBM 3270-BSC Gateway
011	009	IBM 2780-BSC Gateway
012	010	IBM 3790-SDLC Gateway
013	011	TPS application
014	012	RT-11 DIBOL application
015	013	TOPS-20 terminal handler

Object Type		Process Type
Octal	Decimal	
016	014	TOPS–20 remote spooler
017	015	RSX–11M Remote Task Control utility (TCL) Version 2
020	016	TLK utility (LSN)
021	017	File Access Listener (FAL/DAP) Version 4 and later
022	018	RSX–11S Host Loader utility (HLD)
023	019	Network Information and Control Exchange (NICE)
024	020	RSTS/E media transfer program (NETCPY)
025	021	RSTS/E-to-RSTS/E network command terminal handler
026	022	Mail listener (DECnet-based electronic mail system)
027	023	Network command terminal handler (host side)
030	024	Network command terminal handler (terminal side)
031	025	Loopback mirror (MIR)
032	026	Event receiver (EVR)
033	027	VAX/VMS personal message utility
034	028	File Transfer Spooler (FTS)
035	029	PHONE utility
036	030	Distributed data management facility (DDMF)
037	031	X.25 Gateway access
040–076	032–062	Reserved for DECnet use
077	063	DECnet test tool (DTR)
100–177	064–127	Reserved for DECnet use
200–377	128–255	Reserved for customer use

C

Remote File Access Error/Completion Codes

C.1 I/O Status Block Error Returns

Each remote file access subroutine returns a 2-word I/O status block. The contents of the second word depend on the contents of the first word.

Table C–1 describes each code that can be returned in the first word of the status block. The description of the code tells where to look up the description of the value returned in the second word.

Table C–1: First Word I/O Status Block Error Codes

Error Code	Description
177777 (–1)	Channel already active. An attempt was made to open a file on an active channel. Either use another channel or close the active channel before reusing it. The second word of the I/O status block is not applicable.
177776 (–2)	Channel not active A file operation request was made on an inactive channel. Either a file open was not issued on this channel or the network link for this channel was lost. The second word of the I/O status block is not applicable.
177775 (–3)	Data Access Protocol error An error was detected by the remote file system or by the remote server task. DAP then returns the error to the user. The second word of the I/O status block contains the file access error code. Look up this code in Table C–3.

(continued on next page)

Table C–1 (Cont.): First Word I/O Status Block Error Codes

Error Code	Description
177774 (–4)	NSP error (see Table C–2)
	The Data Access Protocol (DAP) utilities use Network Services Protocol (NSP) as a vehicle for accessing remote files. This code indicates that a problem was encountered at the NSP level.
	The low-order byte of the second word of the I/O status block contains one of the NSP error codes listed in Table C–2. If the error is network rejection (–7), the high-order byte of the second word of the I/O status block contains the reject reason code (see Appendix A).
177773 (–5)	Invalid attributes
	An invalid character was found in the attributes array (*ichar*) of an open command.
177772 (–6)	Data overrun
	The received message or block of messages did not fit into the user-specified buffer.
	The second word of the I/O status block contains the total number of bytes read.
177771 (–7)	Tasks out of sync
	The requesting task and its server (FAL) have lost Data Access Protocol (DAP) message synchronization. This indicates a serious internal software problem to report to your system manager.
	The second word of the I/O status block is not applicable.
177770 (–8)	Invalid DAP channel (LUN)
	DAP channel numbers must fall in the range of 1 to 255. A value equal to zero (0) or greater than 255 is invalid.
	The second word of the I/O status block is not applicable.
177767 (–9)	Buffer allocation error for DAP channels
	There is no more buffer space available for the DAP channel control blocks. To extend the buffer size, rebuild the FORTRAN program, increasing the size of $FSR1 in the task build.
	The second word of the I/O status block is not applicable.

Table C–1 (Cont.): First Word I/O Status Block Error Codes

Error Code	Description
177766 (–10.)	Directive error
	Directive error from the executive.
	The second word of the I/O status block contains the DSW value.
177765 (–11.)	Illegal request
	An illegal request was made, such as an attempt to read from a file that was open for write.
	The second word of the I/O status block is not applicable.

Table C–2 contains the NSP error codes that pertain to the NSP error in Table C–1 (177774). NSP error codes occupy the low-order byte of the second word of the I/O status block. With the exception of the network rejection (–7), the high-order byte is undefined.

Table C–2: NSP Error Codes

Error Code	Description
–1	Required system resources are unavailable.
–2	A request was issued for a LUN on which there is no established logical link.
–3	The link was disconnected with the request outstanding.
–4	The data message to be received was truncated because the receive buffer was too small.
–5	An argument specified in the call was incorrect.
–6	No network data was found in the user's mailbox.
–7	The network (NSP) rejected an attempted connect. The high-order byte contains the reject reason code (See Appendix A).
–8	A logical link has already been established on the LUN to which the user attempted to connect.
–9	The issuing task is not part of the network. OPNNT was never called.
–10	The user is attempting to access the network for a second time.
–11	An interrupt message transmission was attempted before the last one had finished.
–12	The user task to which the connection was attempted issued a connect reject.
–13	A buffer is outside the user address space or is not word aligned.
–14	The user is attempting to issue a GNDNT[W] when one is already pending.

C.2 Data Access Protocol (DAP) Error Messages

The DAP status code returns status from the remote file system or from the operation of the cooperating process using DAP. The 2-byte status field (16 bits) occupies the second word of the I/O status block and has two fields:

- Maccode (bits 12–15): Contains the error type code (see Table C–3 in Section C.2.1)
- Miccode (bits 0–11): Contains the specified error reason code (see Tables C–4, C–5, and C–6, depending on error type, as described in Section C.2.2)

C.2.1 Maccode Field

The maccode field is in the high-order byte of the second word in an I/O status block. The value returned in the maccode field describes the functional type of the error. The miccode field gives the specific reason for the error. The miccode field is the low-order byte of the same word that contains the maccode field. The last column of Table C–3 tells which table in this appendix contains the miccode values that correspond to the maccode values.

Table C–3: DAP Maccode Field Values

Field Value (Octal)	Error Type	Meaning	Miccode Table
0	Pending	The operation is in progress.	E–3
1	Successful	Returns information that indicates success.	E–3
2	Unsupported	This implementation of DAP does not support the specified request.	E–2
3	Reserved		
4	File open	Errors that occur before a file is successfully opened.	E–3
5	Transfer error	Errors that occur after a file is opened and before it is closed.	E–3
6	Transfer warning	For operations on open files, indicates that the operation completed, but not with complete success.	E–3
7	Access termination	Errors associated with terminating access to a file.	E–3
10	Format	Error in parsing a message. Format is not correct.	E–2
11	Invalid	Field of message is invalid (that is, bits that are meant to be mutually exclusive are set, an undefined bit is set, a field value is out of range, or an illegal string is in a field.)	E–2
12	Sync	DAP message received out of synchronization.	E–4
13–15	Reserved		
16–17	User-defined status maccodes		

C.2.2 Miccode Field

The miccode field is located in the low-order byte of the second word in an I/O status block. The miccode field value identifies the specific reason for the maccode field error type (see Section C.2.1). Three different tables define the miccode field values, as follows:

- Table C–4: For use with maccode values 2, 10, 11
- Table C–5: For use with maccode values 0, 1, 4, 5, 6, 7
- Table C–6: For use with maccode value 12

Table C–4 follows. The DAP message type number (column 1) is specified in bits 6–11, and the DAP message field number (column 2) is specified in bits 0–5. The third column describes the field where the error is located.

Table C–4: DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Miscellaneous message errors		
00	00	Unspecified DAP message error
	10	DAP message type field (TYPE) error
Configuration message errors		
01	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	BITCNT field (BITCNT)
	20	Buffer size field (BUFSIZ)
	21	Operating system type field (OSTYPE)
	22	File system type field (FILESYS)
	23	DAP version number (VERNUM)
	24	ECO version number field (ECONUM)
	25	USER protocol version number field (USRNUM)
	26	DEC software release number field (DECVER)
	27	User software release number field (USRVER)
	30	System capabilities field (SYSCAP)

(continued on next page)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Attributes message errors		
02	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN 256)
	14	Bit count field (BITCNT)
	20	Attributes menu field (ATTMENU)
	21	Data type field (DATATYPE)
	22	Field organization field (ORG)
	23	Record format field (RFM)
	24	Record attributes field (RAT)
	25	Block size field (BLS)
	26	Maximum record size field (MRS)
	27	Allocation quantity field (ALQ)
	30	Bucket size field (BKS)
	31	Fixed control area size field (FSZ)
	32	Maximum record number field (MRN)
	33	Run-time system field (RUNSYS)
	34	Default extension quantity field (DEQ)
	35	File options field (FOP)
	36	Byte size field (BSZ)
	37	Device characteristics field (DEV)
	40	Spooling device characteristics field (SDC); reserved
	41	Longest record length field (LRL)
	42	Highest virtual block allocated field (HBK)
	43	End-of-file block field (EBK)
	44	First free byte field (FFB)
	45	Starting LBN for contiguous file field (SBN)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Access message errors		
03	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Access function field (ACCFUNC)
	21	Access options field (ACCOPT)
	22	File specification field (FILESPEC)
	23	File access field (FAC)
	24	File-sharing field (SHR)
	25	Display attributes request field (DISPLAY)
	26	File password field (PASSWORD)
Control message errors		
04	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Control function field (CTLFUNC)
	21	Control menu field (CTLMENU)
	22	Record access field (RAC)
	23	Key field (KEY)
	24	Key of reference field (KRF)
	25	Record options field (ROP)
	26	Hash code field (HSH); reserved for future use
	27	Display attributes request field (DISPLAY)
	30	Block count (BLKCNT)
Continue message errors		
05	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Continue transfer function field (CONFUNC)

(continued on next page)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Acknowledge message errors		
06	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	15	System-specific field (SYSPEC)
Access complete message errors		
07	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Access complete function field (CMPFUNC)
	21	File options field (FOP)
	22	Checksum field (CHECK)
Key definition message errors		
12	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Key definition menu field (KEYMENU)
	21	Key option flags field (FLG)
	22	Data bucket fill quantity field (DFL)
	23	Index bucket fill quantity field (IFL)
	24	Key segment repeat count field (SEGCNT)
	25	Key segment position field (POS)
	26	Key segment size field (SIZ)
	27	Key of reference field (REF)
	30	Key name field (KNM)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
	31	Null key character field (NUL)
	32	Index area number field (IAN)
	33	Lowest level area number field (LAN)
	34	Data level area number field (DAN)
	35	Key data type field (DTP)
	36	Root VBN for this key field (RVB)
	37	Hash algorithm value field (HAL)
	40	First data bucket VBN field (DVB)
	41	Data bucket size field (DBS)
	42	Index bucket size field (IBS)
	43	Level of root bucket field (LVL)
	44	Total key size field (TKS)
	45	Minimum record size field (MRL)
Allocation message errors		
13	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Allocation menu field (ALLMENU)
	21	Relative volume number field (VOL)
	22	Alignment options field (ALN)
	23	Allocation options field (AOP)
	24	Starting location field (LOC)
	25	Related file identification field (RFI)
	26	Allocation quantity field (ALQ)
	27	Area identification field (AID)
	30	Bucket size field (BKZ)
	31	Default extension quantity field (DEQ)

(continued on next page)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Summary message errors		
14	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Summary menu field (SUMENU)
	21	Number of keys field (NOK)
	22	Number of areas field (NOA)
	23	Number of record descriptors field (NOR)
	24	Prologue version number (PVN)
Date and time message errors		
15	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Date and time menu field (DATMENU)
	21	Creation date and time field (CDT)
	22	Last update date and time field (RDT)
	23	Deletion date and time field (EDT)
	24	Revision number field (RVN)
	25	Backup date and time field (BDT)
	26	Physical creation date and time field (PDT)
	27	Accessed date and time field (ADT)

Table C–4 (Cont.): DAP Miccode Values for Use with Maccode Values of 2, 10, and 11

Type Number (bits 6–11)	Field Number (bits 0–5)	Field Description
Protection message errors		
16	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Protection menu field (PROTMENU)
	21	File owner field (OWNER)
	22	System protection field (PROTSYS)
	23	Owner protection field (PROTOWN)
	24	Group protection field (PROTGRP)
	25	World protection field (PROTWLD)
Name message errors		
17	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Name type field (NAMETYPE)
	21	Name field (NAMESPEC)
Access control list message errors (reserved for future use)		
20	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	15	System-specific field (SYSPEC)
	20	Access control list repeat count field (ACLCNT)
	21	Access control list entry field (ACL)

Table C–5 follows. The error code number (column 1) is contained in bits 0–11. Symbolic status codes (column 2, when shown) refer to the corresponding RMS or FCS status codes. They are included here for ease of reference only, as they have no meaning for DAP.

Table C–5: DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
0		Unspecified error
1	ER$ABO	Operation aborted
2	ER$ACC	F11ACP could not access file
3	ER$ACT	File activity precludes operation
4	ER$AID	Bad area ID
5	ER$ALN	Alignment options error
6	ER$ALQ	Allocation quantity too large or 0 value
7	ER$ANI	Not ANSI D format
10	ER$AOP	Allocation options error
11	ER$AST	Invalid (synchronous) operation at AST level
12	ER$ATR	Attribute read error
13	ER$ATW	Attribute write error
14	ER$BKS	Bucket size too large
15	ER$BKZ	Bucket size too large
16	ER$BLN	BLN length error
17	ER$BOF	Beginning of file detected
20	ER$BPA	Private pool address
21	ER$BPS	Private pool size
22	ER$BUG	Internal RMS error condition detected
23	ER$CCR	Cannot connect RAB
24	ER$CHG	$UPDATE changed a key without having attribute of XB$CHG set
25	ER$CHK	Bucket format check-byte failure
26	ER$CLS	RSTS/E close function failed
27	ER$COD	Invalid or unsupported COD field
30	ER$CRE	F11ACP could not create file (STV = *system-error-code*)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
31	ER$CUR	No current record (operation not preceded by get/find)
32	ER$DAC	F11ACP deaccess error during close
33	ER$DAN	Data area number invalid
34	ER$DEL	RFA-accessed record was deleted
35	ER$DEV	Bad device, or inappropriate device type
36	ER$DIR	Error in directory name
37	ER$DME	Dynamic memory exhausted
40	ER$DNF	Directory not found
41	ER$DNR	Device not ready
42	ER$DPE	Device has positioning error
43	ER$DTP	DTP field invalid
44	ER$DUP	Duplicate key detected; XB$DUP not set
45	ER$ENT	F11ACP enter function failed
46	ER$ENV	Operation not selected in ORG$ macro
47	ER$EOF	End of file
50	ER$ESS	Expanded string area too short
51	ER$EXP	File expiration date not yet reached
52	ER$EXT	File extend failure
53	ER$FAB	Not a valid FAB (BID does not = FB$BID)
54	ER$FAC	Illegal FAC for record operation, or FB$PUT not set for create
55	ER$FEX	File already exists
56	ER$FID	Invalid file ID
57	ER$FLG	Invalid flag-bits combination
60	ER$FLK	File is locked by other user
61	ER$FND	F11ACP find function failed

(continued on next page)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
62	ER$FNF	File not found
63	ER$FNM	Error in file name
64	ER$FOP	Invalid file options
65	ER$FUL	Device/file full
66	ER$IAN	Index area number invalid
67	ER$IFI	Invalid IFI value or unopened file
70	ER$IMX	Maximum NUM (254) areas/key XABS exceeded
71	ER$INI	$INIT macro never issued
72	ER$IOP	Operation illegal or invalid for file organization
73	ER$IRC	Illegal record encountered (with sequential files only)
74	ER$ISI	Invalid ISI value on unconnected RAB
75	ER$KBF	Bad key buffer address (KBF = 0)
76	ER$KEY	Invalid key field (KEY = 0 or negative)
77	ER$KRF	Invalid key of reference ($GET/$FIND)
100	ER$KSZ	Key size too large
101	ER$LAN	Lowest level index area number invalid
102	ER$LBL	Not ANSI-labeled tape
103	ER$LBY	Logical channel busy
104	ER$LCH	Logical channel number too large
105	ER$LEX	Logical extend error; prior extend still valid
106	ER$LOC	LOC field invalid
107	ER$MAP	Buffer-mapping error
110	ER$MKD	F11ACP could not mark file for deletion
111	ER$MRN	MRN value = negative or relative key > MRN

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
112	ER$MRS	MRS value = 0 for fixed length records and/ or relative files
113	ER$NAM	NAM block address invalid (NAM = 0 or is not accessible)
114	ER$NEF	Not positioned to EOF (with sequential files only)
115	ER$NID	Cannot allocate internal index descriptor
116	ER$NPK	Indexed file; primary key defined
117	ER$OPN	RSTS/E open function failed
120	ER$ORD	XABs not in correct order
121	ER$ORG	Invalid file organization value
122	ER$PLG	Error in file's prologue (reconstruct file)
123	ER$POS	POS field invalid (POS > MRS; STV = XAB indicator)
124	ER$PRM	Bad file date field retrieved
125	ER$PRV	Privilege violation (OS denies access)
126	ER$RAB	Not a valid RAB (BID does not = RB$BID)
127	ER$RAC	Illegal RAC value
130	ER$RAT	Illegal record attributes
131	ER$RBF	Invalid record buffer address (either odd or not word aligned if BLK–IO)
132	ER$RER	File read error
133	ER$REX	Record already exists
134	ER$RFA	Bad RFA value (RFA = 0)
135	ER$RFM	Invalid record format
136	ER$RLK	Target bucket locked by another stream
137	ER$RMV	F11ACP remove function failed
140	ER$RNF	Record not found

(continued on next page)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
141	ER$RNL	Record not locked
142	ER$ROP	Invalid record options
143	ER$RPL	Error while reading prologue
144	ER$RRV	Invalid RRV record encountered
145	ER$RSA	RAB stream currently active
146	ER$RSZ	Bad record size (RSZ > MRS or NOT = MRS if fixed length records)
147	ER$RTB	Record too big for user's buffer
150	ER$SEQ	Primary key out of sequence (RAC = RB$SEQ for $PUT)
151	ER$SHR	SHR field invalid for file (cannot share sequential files)
152	ER$SIZ	SIZ field invalid
153	ER$STK	Stack too big for save area
154	ER$SYS	System directive error
155	ER$TRE	Index tree error
156	ER$TYP	Error in file type (extension on FNS is too big)
157	ER$UBF	Invalid user buffer address (0, odd, or not word aligned if BLK–IO)
160	ER$USZ	Invalid user buffer size (USZ = 0)
161	ER$VER	Error in version number
162	ER$VOL	Invalid volume number
163	ER$WER	File write error (STV = *system-error-code*)
164	ER$WLK	Device is write locked
165	ER$WPL	Error while writing prologue
166	ER$XAB	Not a valid XAB (@XAB = odd; STV = XAB indicator)
167	BUGDDI	Default directory invalid

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
170	CAA	Cannot access argument list
171	CCF	Cannot close file
172	CDA	Cannot deliver AST
173	CHN	Channel assignment failure (STV = *system-error-code*)
174	CNTRLO	Terminal output ignored due to CTRL/O
175	CNTRLY	Terminal input aborted due to CTRL/Y
176	DNA	Default file name string address error
177	DVI	Invalid device ID field
200	ESA	Expanded string address error
201	FNA	File name string address error
202	FSZ	FSZ field invalid
203	IAL	Invalid argument list
204	KFF	Known file found
205	LNE	Logical name error
206	NOD	Node name error
207	NORMAL	Operation successful
210	OK_DUP	Inserted record had duplicate key
211	OK_IDX	Index update error occurred; record inserted
212	OK_RLK	Record locked, but read anyway
213	OK_RRV	Record inserted in primary key is okay; may not be accessible by secondary keys or RFA
214	CREATE	File was created, but not opened
215	PBF	Bad prompt buffer address
216	PNDING	Asynchronous operation pending completion
217	QUO	Quoted string error
220	RHB	Record header buffer invalid

(continued on next page)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
221	RLF	Invalid related file
222	RSS	Invalid resultant string size
223	RST	Invalid resultant string address
224	SQO	Operation not sequential
225	SUC	Operation successful
226	SPRSED	Created file superseded existing version
227	SYN	File name syntax error
230	TMO	Timeout period expired
231	ER$BLK	FB$BLK record attribute not supported
232	ER$BSZ	Bad byte size
233	ER$CDR	Cannot disconnect RAB
234	ER$CGJ	Cannot get JFN for file
235	ER$COF	Cannot open file
236	ER$JFN	Bad JFN value
237	ER$PEF	Cannot position to end of file
240	ER$TRU	Cannot truncate file
241	ER$UDF	File currently in an undefined state; access is denied
242	ER$XCL	File must be opened for exclusive access
243		Directory full
244	IE.HWR	Handler not in system
245	IE.FHE	Fatal hardware error
246		Attempt to write beyond EOF
247	IE.ONP	Hardware option not present
250	IE.DNA	Device not attached
251	IE.DAA	Device already attached

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
252	IE.DUN	Device not attachable
253	IE.RSU	Shared resource in use
254	IE.OVR	Illegal overlay request
255	IE.BCC	Block check or CRC error
256	IE.NOD	Caller's nodes exhausted
257	IE.IFU	Index file full
260	IE.HFU	File header full
261	IE.WAC	Accessed for write
262	IE.CKS	File header checksum failure
263	IE.WAT	Attribute control list error
264	IE.ALN	File already accessed on LUN
265	IE.BTF	Bad tape format
266	IE.ILL	Illegal operation on file descriptor block
267	IE.2DV	Rename; two different devices
270	IE.FEX	Rename; new file name already in use
271	IE.RNM	Cannot rename old file system
272	IE.FOP	File already open
273	IE.VER	Parity error on device
274	IE.EOV	End of volume detected
275	IE.DAO	Data overrun
276	IE.BBE	Bad block on device
277	IE.EOT	End of tape detected
300	IE.NBF	No buffer space for file
301	IE.NBK	File exceeds allocated space; no blocks left
302	IE.NST	Specified task not installed
303	IE.ULK	Unlock error
304	IE.NLN	No file accessed on LUN

(continued on next page)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
305	IE.SRE	Send/receive failure
306	SPL	Spool or submit command file failure
307	NMF	No more files
310	CRC	DAP file transfer checksum error
311		Quota exceeded
312	BUGDAP	Internal network error condition detected
313	CNTRLC	Terminal input aborted due to CTRL/C
314	DFL	Data bucket fill size > bucket size in XAB
315	ESL	Invalid expanded string length
316	IBF	Illegal bucket format
317	IBK	Bucket size of LAN does not = IAN in XAB
320	IDX	Index not initialized
321	IFA	Illegal file attributes (corrupt file header)
322	IFL	Index bucket fill size > bucket size in XAB
323	KNM	Key name buffer cannot be read from or written to in XAB
324	KSI	Index bucket will not hold two keys for key of reference
325	MBC	Multibuffer count invalid (negative value)
326	NET	Network operation failed at remote node
327	OK_ALK	Record is already locked
330	OK_DEL	Deleted record successfully accessed
331	OK_LIM	Retrieved record exceeds specified key value
332	OK_NOP	Key XAB not filled in
333	OK_RNF	Nonexistent record successfully accessed
334	PLV	Unsupported prologue version
335	REF	Illegal key of reference in XAB
336	RSL	Invalid resultant string length
337	RVU	Error updating RRVs; some paths to data may be lost

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
340	SEG	Data types other than string limited to one segment in XAB
341		Reserved
342	SUP	Operation not supported over network
343	WBE	Error on write behind
344	WLD	Invalid wildcard operation
345	WSF	Working set full (cannot lock buffers in working set)
346		Directory listing: error in reading volume set name, directory name, or file name
347		Directory listing: error in reading file attributes
350		Directory listing: protection violation in trying to read the volume set, directory, or file name
351		Directory listing: protection violation in trying to read file attributes
352		Directory listing: file attributes do not exist
353		Directory listing: unable to recover directory list after continue transfer (skip)
354	SNE	Sharing not enabled
355	SPE	Sharing page count exceeded
356	UPI	UPI bit not set when sharing with BRO set
357	ACS	Error in access control string
360	TNS	Terminator not seen
361	BES	Bad escape sequence
362	PES	Partial escape sequence
363	WCC	Invalid wildcard context value
364	IDR	Invalid directory rename operation

(continued on next page)

Table C–5 (Cont.): DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

Error Code (bits 0–11)	Symbolic Status Code	Error Description
365	STR	User structure (FAB/RAB) became invalid during operation
366	FTM	Network file transfer mode precludes operation
6000 to 7777		User-defined errors

Table C–6 follows. The message type number is contained in bits 0–11.

Table C–6: DAP Miccode Values for Use with Maccode Value 12

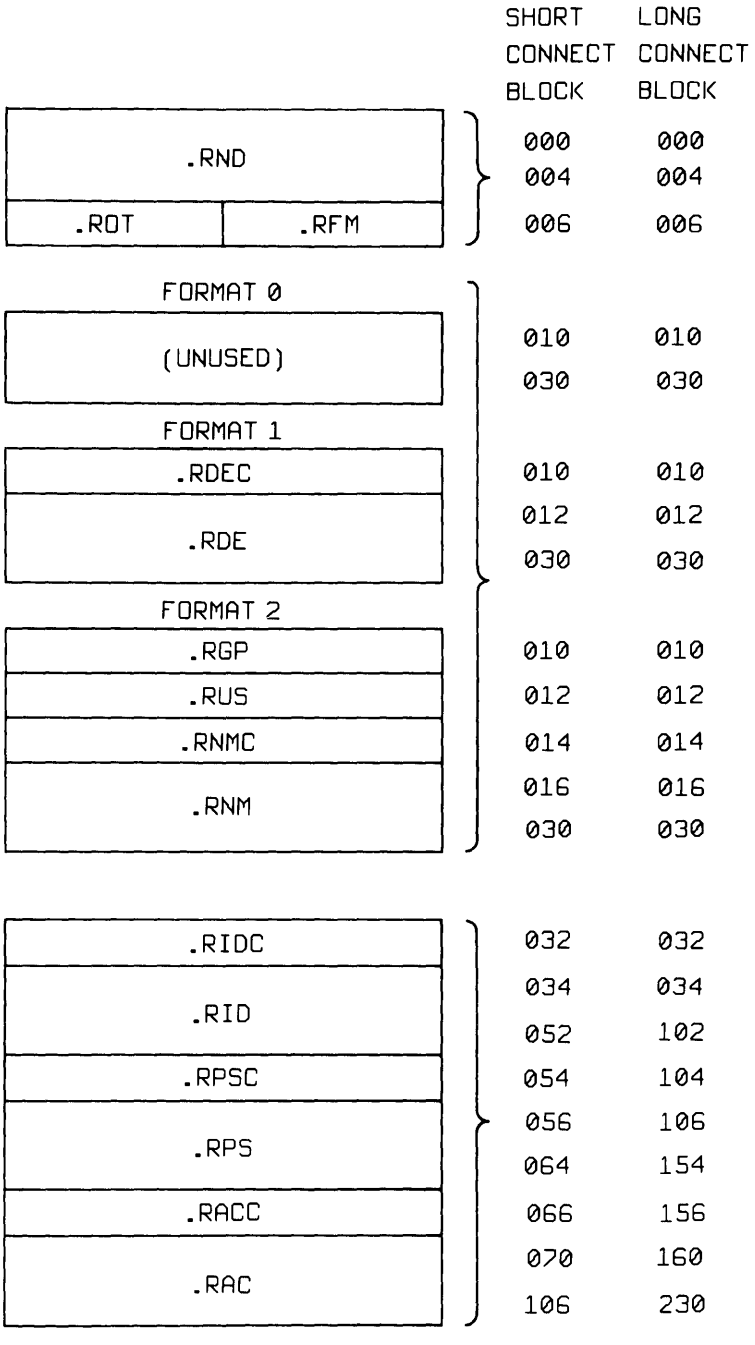
Type Number (bits 0–11)	Message Type
0	Unknown message type
1	Configuration message
2	Attributes message
3	Access message
4	Control message
5	Continue transfer message
6	Acknowledge message
7	Access complete message
10	Data message
11	Status message
12	Key definition attributes extension message
13	Allocation attributes extension message
14	Summary attributes extension message
15	Date and time attributes extension message
16	Protection attributes extension message
17	Name message
20	Access control list extended attributes message

D

MACRO–11 Connect Block Offset and Code Definitions

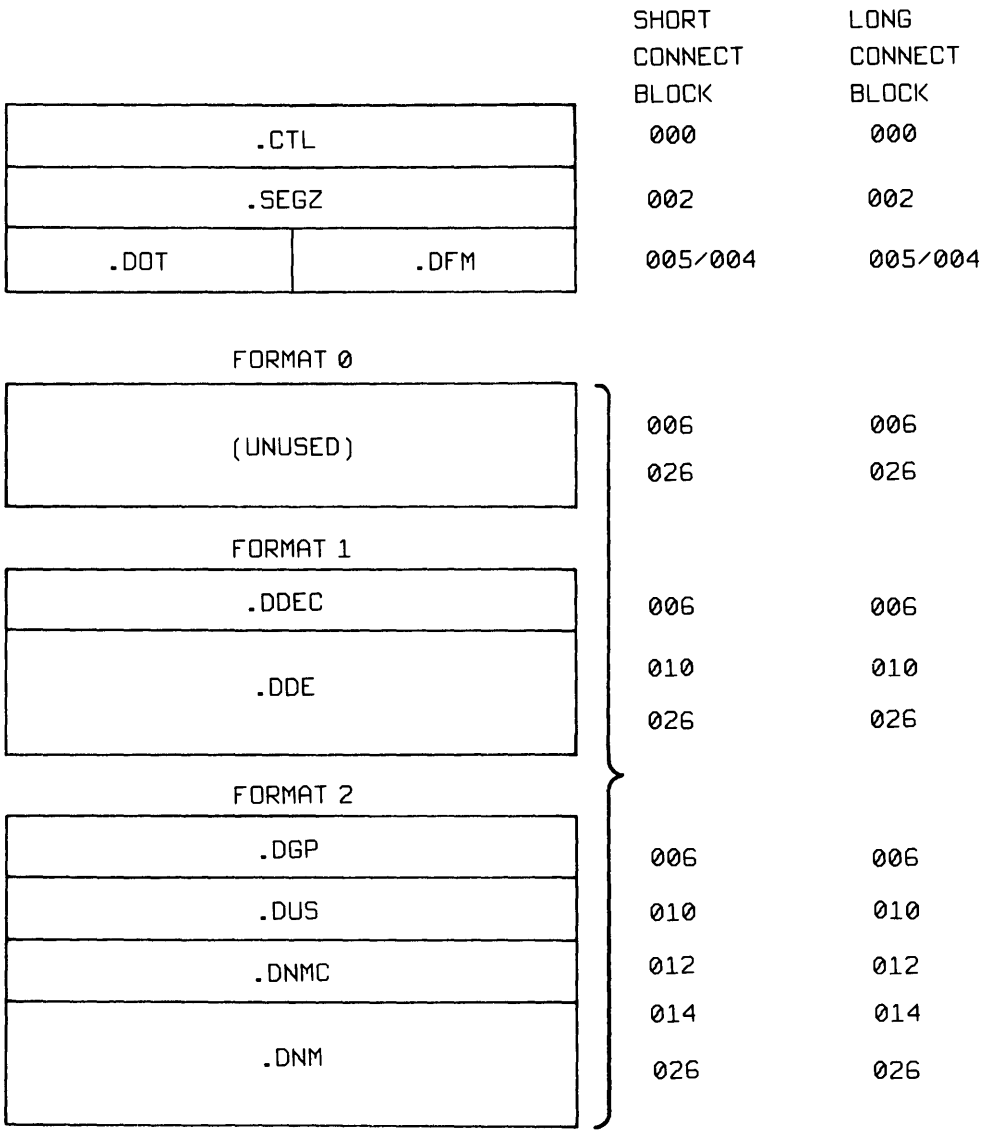
The following MACRO–11 offset and code definitions refer to connect block offsets used in network connects and accepts.

```
        .TITLE NETDEF - DECnet User Interface Definitions
        .IDENT /V02.05/
;
; Copyright (C) 1978, 1979, 1980, 1987 by
; Digital Equipment Corporation, Maynard, Mass.
;
; Module Description:
;
;       RSX-11M/S/M-PLUS Network Interface Offset and Error Definitions
;+
; Macro to define DECnet MACRO user interface data structures
; and return values.
;-

        .MACRO NETDF$,L,B

        .MCALL CRBDF$
        CRBDF$ L,B              ; Request descriptor block
        .MCALL CNBDF$
        CNBDF$ L,B              ; Request pending block
        .MCALL NSSYM$
        NSSYM$ B                ; Return symbols

        .MACRO NETDF$,X,Y
        .ENDM NETDF$

        .ENDM NETDF$
;+
; Request Descriptor Block offset definitions for connects.
;
; NOTE: Long connect block offsets are prefixed with M.
;       Short connect block offsets are prefixed with N.
;
; The figure does not include the offset prefix.  If you are using
; the long connect block, use M for the offset prefix.  If you are
; using the short connect block, use N.
```


SHORT CONNECT BLOCK
LONG CONNECT BLOCK
.RND
.ROT
.RFM
000 000
004 004
006 006
FORMAT 0
(UNUSED)
010 010
030 030
FORMAT 1
.RDEC
.RDE
010 010
012 012
030 030
FORMAT 2
.RGP
.RUS
.RNMC
.RNM
010 010
012 012
014 014
016 016
030 030
.RIDC
.RID
.RPSC
.RPS
.RACC
.RAC
032 032
034 034
052 102
054 104
056 106
064 154
066 156
070 160
106 230

```
        .MACRO  CRBDF$,L,B,LST
        .iif nb LST      .List
        .ASECT
;
; Long connect block offsets (support for long passwords, user names and
; accounting information)
;
.=0
M.RND:'L'    .BLKB        6      ; Destination node name
M.RFM:'L'    .BLKB        1      ; Destination descriptor format
M.ROT:'L'    .BLKB        1      ; Destination object type

                          ;******
                          ;              Format 0 -
        .BLKB    18.      ; [UNUSED]
                          ;***
.=.-18.                   ;              Format 1 -
M.RDEC:'L'   .BLKW        1      ; Destination process byte count
M.RDE:'L'    .BLKB        16.    ; Destination process
                          ;***
;
; Offsets between N.RND and N.RDE and M.RND and M.RDE for format 1 or N.RGP
; and M.RGP  for format 2 must be identical in both the short and long
; connect blocks.  Do not add an offset to one without adding it to the other.
;
.=.-18.                   ;                Format 2 -
M.RGP:'L'   .BLKW         1      ; Destination group
M.RUS:'L'   .BLKW         1      ; Destination user
M.RNMC:'L' .BLKW          1      ; Destination name byte count
M.RNM:'L'   .BLKB         12.    ; Destination name
                          ;******

M.RIDC:'L' .BLKW          1      ; Requesting process ID byte count
M.RID:'L'   .BLKB         40.    ; Requesting process ID
M.RPSC:'L' .BLKW          1      ; Requesting password byte count
M.RPS:'L'   .BLKB         40.    ; Requesting password byte count
M.RACC:'L' .BLKW          1      ; Accounting information byte count
M.RAC:'L'   .BLKB         40.    ; Accounting information
                          ;
M.RQL='B'.-M.RND                 ; Length of RDB
;
;
; Short connect block offsets.  Included for compatibility with
; existing software and versions of DECnet RSX
;
;
```

```
.=0
N.RND:'L'  .BLKB        6     ; Destination node name
N.RFM:'L'  .BLKB        1     ; Destination descriptor format
N.ROT:'L'  .BLKB        1     ; Destination object type

                        ;******
                        ;               Format 0 -
       .BLKB   18.      ; [UNUSED]
                        ;***
.=.-18.                 ;               Format 1 -
N.RDEC:'L' .BLKW        1     ; Destination process byte count
N.RDE:'L'  .BLKB        16.   ; Destination process
                        ;***
;
; Offsets between N.RND and N.RDE and M.RND and M.RDE for format 1 or N.RGP
; and M.RGP  for format 2 must be identical in both the short and long
; connect blocks.  Do not add an offset to one without adding it to the other.
;
;
.=.-18.                 ;               Format 2 -
N.RGP:'L'  .BLKW        1     ; Destination group
N.RUS:'L'  .BLKW        1     ; Destination user
N.RNMC:'L' .BLKW        1     ; Destination name byte count
N.RNM:'L'  .BLKB        12.   ; Destination name
                        ;******

N.RIDC:'L' .BLKW        1     ; Requesting process ID byte count
N.RID:'L'  .BLKB        16.   ; Requesting process id
N.RPSC:'L' .BLKW        1     ; Requesting password byte count
N.RPS:'L'  .BLKB        8.    ; Requesting password
N.RACC:'L' .BLKW        1     ; Accounting information byte count
N.RAC:'L'  .BLKB        16.   ; Accounting information
                        ;
N.RQL='B'.-N.RND              ; Length of short RDB

        .PSECT

        .if nb LST
        .Nlist
        .iff
        .MACRO CRBDF$,X,Y,Z
        .ENDM  CRBDF$
        .endc

        .ENDM CRBDF$
;+
; Connect Block offset definitions for received connect requests.
;
; NOTE: Long connect block offsets are prefixed with M  (example: M.RND)
;       Short connect block offsets are prefixed with N.
;
; The offset prefix is not included in the diagram.  If you are using
; the long connect block, use M for the offset prefix.  If you are
; using the short connect block, use N.
```


SHORT CONNECT BLOCK
LONG CONNECT BLOCK
.CTL
000
000
.SEGZ
002
002
.DOT
.DFM
005/004
005/004
FORMAT 0
(UNUSED)
006
026
006
026
FORMAT 1
.DDEC
.DDE
006
010
026
006
010
026
FORMAT 2
.DGP
.DUS
.DNMC
.DNM
006
010
012
014
026
006
010
012
014
026

```
      .MACRO CNBDF$,L,B,LST
      .iif nb LST .List
      .ASECT
;
; Incoming short connect block offsets
;
.=0
M.CTL:'L'   .BLKW       1     ; Temporary link address
                        ;
M.SEGZ:'L'  .BLKW       1     ; Segment size
M.DFM:'L'   .BLKB       1     ; Destination descriptor format
M.DOT:'L'   .BLKB       1     ; Destination object type

                        ;******
                        ;     Format 0 -
        .BLKB    18.    ; [UNUSED]
                        ;***
.=.-18.                 ;     Format 1 -
M.DDEC:'L'  .BLKW       1     ; Destination process byte count
M.DDE:'L'   .BLKB       16.   ; Destination process
                        ;***
.=.-18.                 ;     Format 2 -
M.DGP:'L'   .BLKW       1     ; Destination group
M.DUS:'L'   .BLKW       1     ; Destination user
M.DNMC:'L'  .BLKW       1     ; Destination name byte count
M.DNM:'L'   .BLKB       12.   ; Destination name
                        ;******

M.SND:'L'   .BLKB       6     ; Source node name
M.SFM:'L'   .BLKB       1     ; Source descriptor format
M.SOT:'L'   .BLKB       1     ; Source object type

                        ;******
                        ;     Format 0 -
        .BLKB     18.   ; [UNUSED]
                        ;***
.=.-18.                 ;     Format 1 -
M.SDEC:'L'  .BLKW       1     ; Source process name byte count
M.SDE:'L'   .BLKB       16.   ; Source process name
                        ;***
.=.-18.                 ;     Format 2 -
M.SGP:'L'   .BLKW       1     ; Source group
M.SUS:'L'   .BLKW       1     ; Source user
M.SNMC:'L'  .BLKW       1     ; Source name byte count
M.SNM:'L'   .BLKB       12.   ; Source name
                        ;******
```

```
$$$=.
M.CIDC:'L'  .BLKW       1     ; Source task ID byte count
M.CID:'L'   .BLKB       40.   ; Source task ID
M.CPSC:'L'  .BLKW       1     ; Password byte count
M.CPS:'L'   .BLKB       40.   ; Password
M.CACC:'L'  .BLKW       1     ; Accounting information byte count
M.CAC:'L'   .BLKB       40.   ; Accounting information
M.CDAC:'L'  .BLKW       1     ; Optional data byte count
M.CDA:'L'                     ; Optional data buffer
                        ;
M.CBL='B'.-M.CTL              ; Length of CNB (without any data)

.=$$$
M.CDEV:'L'  .BLKW       1     ; Default device name (from account file)
M.CUNI:'L'  .BLKB       1     ; Default device unit number
      .EVEN
M.CUIC:'L'  .BLKW       1     ; Login UIC from account file
M.CDDS:'L'  .BLKB       11.   ; Default directory string (byte 0=0 => none)
;
;  Incoming short connect block offsets
;
.=0
N.CTL:'L'   .BLKW       1    ; Temporary link address
                        ;
N.SEGZ:'L'  .BLKW       1    ; Segment size
N.DFM:'L'   .BLKB       1    ; Destination descriptor format
N.DOT:'L'   .BLKB       1    ; Destination object type

                        ;******
                        ;     Format 0 -
        .BLKB     18.   ; [UNUSED]
                        ;***
.=.-18.                 ;    Format 1 -
N.DDEC:'L'  .BLKW       1    ; Destination process byte count
N.DDE:'L'   .BLKB       16.  ; Destination process
                        ;***
.=.-18.                 ;    Format 2 -
N.DGP:'L'   .BLKW       1    ; Destination group
N.DUS:'L'   .BLKW       1    ; Destination user
N.DNMC:'L'  .BLKW       1    ; Destination name byte count
N.DNM:'L'   .BLKB       12.  ; Destination name
                       ;******

N.SND:'L'   .BLKB       6    ; Source node name
N.SFM:'L'   .BLKB       1    ; Source descriptor format
N.SOT:'L'   .BLKB       1    ; Source object type
```

```
                        ;******
                        ;     Format 0 -
        .BLKB     18.   ; [UNUSED]
                        ;***
.=.-18.                 ;     Format 1 -
N.SDEC:'L'  .BLKW       1     ; Source process name byte count
N.SDE:'L'   .BLKB       16.   ; Source process name
                        ;***
.=.-18.                 ;     Format 2 -
N.SGP:'L'   .BLKW       1     ; Source group
N.SUS:'L'   .BLKW       1     ; Source user
N.SNMC:'L'  .BLKW       1     ; Source name byte count
N.SNM:'L'   .BLKB       12.   ; Source name
                        ;******

$$$=.
N.CIDC:'L'  .BLKW       1     ; Source task ID byte count
N.CID:'L'   .BLKB       16.   ; Source task ID
N.CPSC:'L'  .BLKW       1     ; Password byte count
N.CPS:'L'   .BLKB       8.    ; Password
N.CACC:'L'  .BLKW       1     ; Accounting information byte count
N.CAC:'L'   .BLKB       16.   ; Accounting information
N.CDAC:'L'  .BLKW       1     ; Optional data byte count
N.CDA:'L'                     ; Optional data buffer
                        ;
N.CBL='B'.-N.CTL              ; Length of CNB (without any data)

.=$$$
N.CDEV:'L'  .BLKW       1     ; Default device name (from account file)
N.CUNI:'L'  .BLKB       1     ; Default device unit number
        .EVEN
N.CUIC:'L'  .BLKW       1     ; Login UIC from account file
N.CDDS:'L'  .BLKB       11    ; Default directory string (byte 0=0 => none)

        .PSECT
        .ENDM    CNBDF$
```

E

Network Error/Completion Codes for FORTRAN, COBOL, and BASIC–PLUS–2

This appendix lists the error/completion codes that can be returned in the first word of any 2-word I/O status block by certain calls in the FORTRAN, COBOL, and BASIC–PLUS–2 languages.

1	The request was successful.
2	The request was successful, but some optional data was lost.
–1	Required system resources are not available.
–2	A request was issued for a LUN on which there is no established logical link.
–3	The link was disconnected with the request outstanding.
–4	The data received was truncated because the receive buffer was too small.
–5	An argument specified in the call is incorrect.
–6	No network data was found in the user's network data queue.
–7	The network (NSP) rejected an attempted connect.
–8	A logical link has already been established on the LUN to which the user attempted to connect.
–9	The issuing task is not part of the network (that is, OPNNT was never called).

-10	The user is attempting to access the network for a second time.
-11	Transmission of an interrupt message was attempted before the last one finished.
-12	A connect reject was issued by the user task to which the connection was attempted.
-13	A buffer either is outside the user address space or is not word aligned.
-14	The user is attempting to issue a GNDNT[W] when one is already pending.
-20	A RUNNCW was issued for which there was not enough dynamic memory on the remote node.
-21	A RUNNCW or ABONCW was issued for a task that was not installed on the remote node.
-22	A RUNNCW was issued with an invalid time parameter.
-23	Either an ABONCW was issued for a task that was not active, or a RUNNCW without scheduling parameters was issued for a task that already is active.
-24	There was a privilege violation on an RUNNCW or ABONCW attempt.
-25	An ABONCW was issued for a task that either was being loaded into or was exiting from the remote node.
-26	An RUNNCW was issued with an invalid UIC.
-40	A directive error; the second word of the status block contains the actual directive error code.

F

Network MACRO-11 Error/Completion Codes

Applicable Standard RSX Codes

The following MACRO-11 error completion codes include all network related I/O error completion codes for this manual. These codes are defined in the IOERR$ macro in RSXMAC.SML, which is referenced in the NSSYM$ macro in NETLIB.MLB.

Symbol Name	Decimal Value	Octal Value	Meaning
IS.SUC	1	1	The request was successful.
IS.DAO	2	2	The request was successful, but some data was lost.
IE.BAD	-1	377	Invalid buffer parameter, or data length exceeds 16. bytes.
IE.SPC	-6	372	Invalid buffer parameters: the buffer may not be word-aligned, may be outside user address space, or may exceed 8128. bytes.
IE.WLK	-12	364	Transmission of an interrupt message was attempted before the last one finished.
IE.DAO	-13	363	Data overrun; unstored data was lost.
IE.ABO	-15	361	The link was aborted or disconnected (see disconnect and reject reason codes, Appendix A).

Symbol Name	Decimal Value	Octal Value	Meaning
IE.PRI	-16	360	The network is not accessed on this LUN.
IE.RSU	-17	357	Required system resources are not available.
IE.ALN	-34	336	The specified LUN is already established.
IE.NLN	-37	333	There is no established logical link on the specified LUN.
IE.URJ	-73	267	The remote task rejected an attempted connection.
IE.NRJ	-74	266	The network rejected an attempted connection (see disconnect and reject reason codes, Appendix A).
IE.NDA	-78	262	There is no data to return.
IE.NNT	-94	242	The issuing task is not a network task; OPN$ was not executed successfully.

G

Values for Ethernet and 802.3 Addressing

This appendix provides information on assigned values for

- Multicast addresses
- Protocol types for Ethernet format
- Service Access Point (SAP) addresses for 802.3 format
- Subnetwork Access Protocol (SNAP) identifiers for 802.3 format

All values are in hexadecimal notation, with *nn* representing variables in addresses and protocols.

Note that the protocols and addresses for customer use will not change, but the assigned cross-company and internal Digital values may increase beyond the list in this appendix. The IEEE is continuing to assign SAP and SNAP values; the 802.3 information is currently valid but is still changing.

While this appendix includes the assigned multicast and SAP values for broadcasts, you should avoid using them, since broadcasting congests the network.

G.1 Multicast Addresses

Multicast addresses in the following format are reserved for Digital Equipment Corporation customer use:

AB-00-04-00-*nn-nn*

All companies can use the following reserved multicast addresses:

Value	Meaning
FF–FF–FF–FF–FF–FF	Broadcast
CF–00–00–00–00–00	Loopback assistance

Digital Equipment Corporation reserves the following ranges for internal use only:

08–00–2B–*nn-nn-nn*

09–00–2B–*nn-nn-nn*

AA–00–00–*nn-nn-nn*
AA–00–01–*nn-nn-nn*
AA–00–02–*nn-nn-nn*
AA–00–03–*nn-nn-nn*
AA–00–04–*nn-nn-nn*

AB–00–00–01–00–00
AB–00–00–02–00–00
AB–00–00–03–00–00
AB–00–00–04–00–00

AB–00–04–01–*nn-nn*

In a Digital-only environment, you can use any values that fall outside of the Digital Equipment Corporation ranges. In a multi-vendor environment, however, these values might conflict with the system software of the other vendors. Addresses in the Digital customer range, in contrast, are reserved; they will not conflict even in a multi-vendor environment.

G.1.1 Ethernet Protocol Types

The following protocol type is reserved for Digital Equipment Corporation customer use:

60–06

All companies using the Ethernet protocol use the following protocol type:

Value	Meaning
90–00	Loopback

Digital Equipment Corporation reserves protocol types in the following ranges for internal use only:

60–00 to 60–05

60–07 to 60–09

80–38 to 80–42

In a Digital-only environment, you can use values outside of the Digital Equipment Corporation range. In a multi-vendor environment, however, using other values could cause conflicts with other vendors' software.

Values in the range 00–00 through 05–DC are reserved for internal use and will cause data link level errors in an application.

G.2 SAP Addresses

The IEEE has not yet assigned SAPs to Digital Equipment Corporation, but it has assigned the following for inter-company use:

Value	Name/Meaning
00	The null SAP: addresses just the data link layer, as in an XID or a TEST message.
02	The logical link control sub-layer management function individual SAP: addresses an individual network management entity on the system.
03	The logical link control sub-layer management function group SAP: addresses to all network management entities on the system.
AA	The SNAP SAP: indicates that the next five bytes of a UI frame contain a SNAP identifier.
FF	The global SAP: broadcasts to all 802 receivers on a node.

The IEEE has not yet assigned SAPs for Digital internal or customer use.

G.3 SNAP Identifiers

The following SNAP identifier is reserved for Digital customer use:

08-00-2B-60-06

Digital reserves the following SNAP identifiers for internal use only:

08-00-2B-60-*nn*
08-00-2B-80-3C
08-00-2B-80-3E

H

DLX Characteristics Status Codes

This appendix lists the status codes for DLX characteristics and describes the conditions that return each code, noting any conditions unique to specific QIOs or characteristics blocks.

Error codes have the prefix CE. and a negative 16-bit value. Full or partial success codes have the prefix CS. and a positive 16-bit value. CS.SUC (1) is the code for complete success; other CS. codes indicate partial success that may return unexpected results.

Table H–1: Status Codes for DLX Characteristics

CE. Codes			
Status Code	**Meaning**	**Characteristic**	**Explanation**
CE.ACN 100012	Address conflict	CC.MCT	The protocol/address pair is already in use.
CE.FMI 100015	Frame format invalid	CC.FMO or CC.FMM	You entered an invalid frame format value. The value must be NX$ETH or NX$802.
CE.FMC 100016	Frame usage conflict	CC.FMO or CC.FRM	You specified both frame formats, but only one is valid.
		CC.GSP, CC.SCO, CC.ISP, CC.SNP	The port is not enabled for 802.3 format.
		CC.GSP	The port is not enabled for 802.3 format with Class I service.

(continued on next page)

Table H–1 (Cont.): Status Codes for DLX Characteristics

CE. Codes (Cont.)

Status Code	Meaning	Characteristic	Explanation
CE.IUN 100013	Invalid use of multicast address	CC.DST	You specified a multicast instead of a physical address.
CE.MCE 100007	Multicast address enabled	CC.MCT	The specified multicast address is already enabled.
CE.NMA 100014	Not a multicast address	CC.MCT	The multicast address you entered is not valid. Check that the least significant bit is 1.
CE.PCN 100011	Protocol conflict	CC.DST	In attempting to enable a protocol, your application: ▪ Attempted to enable itself as the default protocol user, but a default user already exists. ▪ Attempted to enable a protocol but an exclusive user for the protocol already exists. ▪ Attempted to enable an already-enabled protocol with a padding status that conflicts with its current status. The first request to enable a protocol type assigns the padding status to which all subsequent uses of the protocol type conform.
CE.RES 100010	Resource allocation failure	Various	No memory is available for the characteristics operation.
CE.RTS 100004	Request too small	Various, with IO.XGC	You allocated too little space for the returned data.
		Various, with IO.XSC	You supplied too little data.
CE.RTL 100003	Request too large	Various, with IO.XGC	You allocated too much space for the returned data.
		Various, with IO.XSC	You supplied too much data for the allocated space.

Table H–1 (Cont.): Status Codes for DLX Characteristics

Status Code	Meaning	Characteristic	Explanation
CE. Codes (Cont.)			
CE.SNU 100021	SNAP in use	CC.SNP	Another port has already enabled the specified SNAP protocol identifier.
CE.SPU 100020	SAP in use	CC.ISP	Another port has already enabled the specified SAP.
CE.SRI 100017	Service class invalid	CC.SCO, CC.SRV	You entered a value other than PF$CLI (10).
CE.UDF 100001	Undefined function	All	The value in the C.TYPE field does not identify a valid characteristic type.
CS. Codes			
CS.DAO 000003	Data overrun	CC.DST with IO.XGC	Returned information exceeded allocated space. Ethernet protocol type information included more addresses than you allocated space for.
		CC.SNP with IO.XGC	SNAP protocol identifier information included more addresses than you allocated space for.
CS.IGN 000002	Ignored	All	This code generally indicates that characteristics information was inappropriate. The code indicates various errors, including: your existing environment is incompatible with the characteristic type (in frame format, for example); the data you supplied was incomplete or incorrect for the characteristic; you already specified the characteristic and this is a redundant block.
CS.SUC 1	Success	All	The characteristics block processed successfully.

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

W

X

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
and Puerto Rico
call 800–258–1710

In Canada
call 800–267–6146

In New Hampshire
Alaska or Hawaii
call 603–884–6660

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575

DECnet–RSX
Programmer's Reference Manual
AA–M098D–TC

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip Code _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE