Networks • Communications

DECnet-RSX

Programmer's Reference Manual

digital software

DECnet-RSX

Programmer's Reference Manual

Order No. AA-M098C-TC

September 1985

The *DECnet-RSX Programmer's Reference Manual* describes the programming facilities available in the DECnet-RSX environment and details the network programming calls for each facility.

Supersession/Update Information:	This is a new manual.
Operating System and Version:	RSX-11M V4.2 RSX-11S V4.2 RSX-11M-PLUS V3.0 Micro/RSX V3.0
Software Version:	DECnet-11M V4.2 DECnet-11S V4.2 DECnet-11M-PLUS V3.0 DECnet-Micro/RSX V1.0

digital

AA-M098C-TC
First Printing, September 1985

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.

Networks and Communications Publications typeset this manual using Digital's TMS-11 Text Management System.

Contents

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

Figures

Tables

Preface

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

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

DECnet-RSX 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.

Intended Audience

This manual is intended for users responsible for writing network programs to be run on DECnet-11M, DECnet-11M-PLUS, DECnet-11S, and DECnet-Micro/RSX systems.

Structure of This Manual

This manual is organized into four chapters:

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 Controller programming facilities for tasks using Ethernet and non-Ethernet QIO calls.

The following programming examples are included in this manual and in your tape or disk kit:

- MACRO-11 Intertask Communication Programming Example (Transmit)
- MACRO-11 Intertask Communication Programming Example (Receive)
- FORTRAN Intertask Communication Programming Example (Transmit)
- FORTRAN Intertask Communication Programming Example (Receive)
- COBOL Intertask Communication Programming Example (Transmit)
- COBOL Intertask Communication Programming Example (Receive)
- BASIC-PLUS-2 Intertask Communication Programming Example (Transmit)
- BASIC-PLUS-2 Intertask Communication Programming Example (Receive)
- FORTRAN Remote File Access Programming Example (Append)
- FORTRAN Remote File Access Programming Example (Read/Write)
- COBOL Remote File Access Programming Example (Append)
- COBOL Remote File Access Programming Example (Read/Write)
- BASIC-PLUS-2 Remote File Access Programming Example (Append)
- BASIC-PLUS-2 Remote File Access Programming Example (Read/Write)
- FORTRAN Task Control Programming Example
- DLX QIO Transmit Programming Example (for non-Ethernet device)
- DLX QIO Receiver Programming Example (for non-Ethernet device)
- DLX QIO Programming Example (for Ethernet device)

This manual also contains the following appendixes:

Appendix A Contains the network disconnect or reject reason codes.

Appendix B Defines the Digital object type code values.

Appendix C Provides a summary of 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.

Associated Documents

Users of this manual should have the following Digital documents available for reference:

- *Introduction to DECnet*
- *DECnet–RSX Network Management Concepts and Procedures*
- *DECnet–RSX Guide to Network Management Utilities*
- *DECnet–RSX Guide to User Utilities*

Users of this manual should also have the RSX–11M documentation set and the appropriate language manuals.

Graphic Conventions

In addition to the graphic conventions listed below, this manual contains specific conventions not defined here. Chapter 2 provides specific conventions for MACRO–11 calls. Chapter 3 provides specific conventions for FORTRAN, COBOL, and BASIC–PLUS–2 calls. See Sections 2.2 and 3.6 for these conventions.

CALL CLSNFW	Capital letters in a command line represent characters that must be typed as shown.
(*lun,status*)	Italicized lowercase letters represent variables for which you must supply specific information.
PRSIZ=5 . . . CALL BFMT1	The use of ellipses means that not all the information the system would display in response to a particular command or message is shown or that not all the information a user would enter is shown.
CTRL/*x*	The expression CTRL/*x* refers to a control character keying sequence. Press the key labeled CTRL and the appropriate character key simultaneously when you see this symbol.

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

1
Introduction

DECnet–RSX software extends the RSX–11M/11M–PLUS/11S operating systems for the PDP–11. With DECnet–RSX you can write programs that exchange data with programs running on other DECnet systems in the network, even though these systems may run under operating systems other than RSX. This manual describes the network programming functions available using the MACRO–11, FORTRAN 77, COBOL, and BASIC–PLUS–2 languages.

Before proceeding, you should become familiar with the *Introduction to DECnet* for background information. Also refer to the *DECnet–RSX Guide to User Utilities* for terminal user information.

For programs using DECnet–RSX, tasks running on different nodes can 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. The DECnet software allows you to perform task-to-task communication regardless of the programming language used or the operating system running on the different nodes. For example, an RSX FORTRAN–77 program can communicate with a VMS program written in BASIC–PLUS–2.

- **Remote file access.** Remote file access programs allow you to access sequential files for reading, writing, and appending records to remote files. User tasks can delete files from devices on remote nodes as well.

  When the remote node is an RSX node, programming remote file access operations is similar to programming local I/O operations. When the remote node is other than an RSX node, an accessing program can perform only those functions that its source language provides and that the remote file system supports. Refer to the *Introduction to DECnet* for the possible file operations for different DECnet systems.

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

Functions performed by cooperating local and remote nodes can also be performed 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 allows programmers to debug RSX programs locally before running them at a remote node.

1.1 Intertask Communication Conventions

This manual refers to calls by their first three letters. These calls are common to all four languages DECnet-RSX supports. The term macro refers to MACRO-11 calls, and the term call refers to the higher level languages. Each call name concludes with a lowercase *x*, which represents the variable portion of each call (determined by the programming language used). When you issue a call in MACRO-11, replace the variable *x* with a $. When you issue a call in FORTRAN, COBOL, or BASIC-PLUS-2, replace the *x* with NT. Lowercase *x* is used throughout this chapter to discuss intertask communication calls in a general way.

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, at the end of Section 1.2.10, provides an alphabetical list of the DECnet intertask communication calls, gives the function of each call, and briefly describes the normal or expected result after executing each call.

1.2 Intertask Communication Concepts

Intertask communication concepts include:

- 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, the task must be an active network task. A task is active if it is running and it has issued an open (OPN*x*) call (see Section 1.1). An OPN*x* call establishes a network data queue for a task and connects the task to the network.

1.2.2 Assigning Logical Unit Numbers

The following calls use a logical unit number (LUN) assigned to the network data queue:

OPN$ Access the network

SPA$ Specify a user AST routine

GND$ Get network data

REJ$ Reject a logical link request

CLS$ End a task's network operations

GLN$ Get local node information

You can assign the LUN by defining it either as the global symbol .MBXLU in your program or as a parameter for the macro call. The .MBXLU definition and the macro call parameter definition are mutually exclusive (.MBXLU is referenced only if the LUN argument is left blank in the macro parameter block). You should define a particular network data queue LUN in one place only.

If you use the .MBXLU definition, you can assign the LUN at assembly time or at task build time using one of the following command techniques:

1. To include the logical unit number (LUN) locally in your source code, include the following in each source module:

   .MBXLU=*x*

   The variable *x* is an integer representing the logical unit number.

2. Each source module in the user task must have the same integer *x* defined for .MBXLU; otherwise, the macros will complete with a privilege error (IE.PRI). You can define a global definition (==) by including the following statement in a single source module:

   .MBXLU==*x*

   This statement causes the task builder to define references to .MBXLU in all modules of your program to the value of *x*.

3. If you want to defer definition of .MBXLU to task build time, you can issue the following task build option:

   GBLDEF=.MBXLU:*x*

   This option instructs the task builder to define all global references to .MBXLU as the value of *x*.

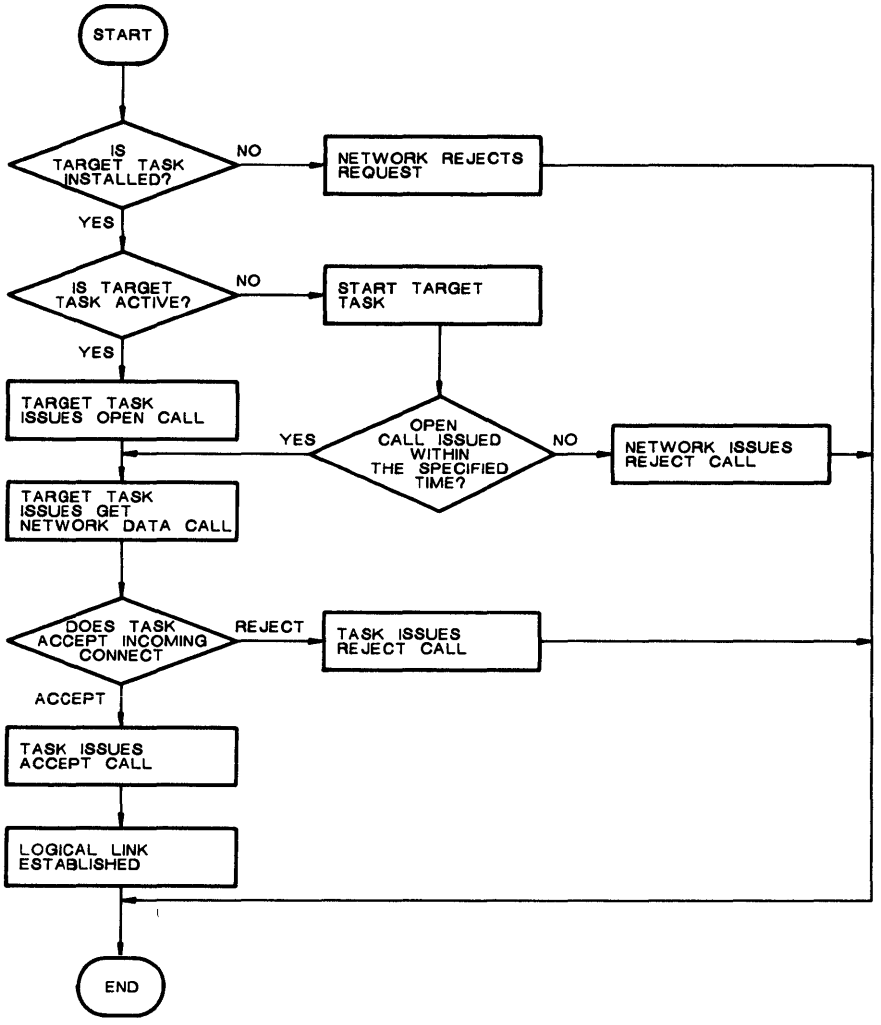


Figure 1–1: Establishing a Logical Link

These three command techniques for defining .MBXLU are mutually exclusive. If you do not use any of these procedures, the task builder returns an undefined reference warning message. If you run the task and ignore the warning, the six macro calls (OPN$, SPA$, GND$, REJ$, CLS$, and GLN$) will be rejected by the operating system with a directive status error indicating an invalid logical unit number. The task builder causes undefined references to default to 0. .MBXLU cannot be defined with a 0 value because 0 is an invalid logical unit number.

All network logical unit numbers are freed and the task's logical links are aborted when you issue a CLS$ call to terminate network operations for a MACRO–11 task. The CLS$ call can be issued 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 will be returned with a status of illegal function code.

1.2.3 Establishing a Logical Link

To exchange data, a logical link must be established between two active network tasks. A logical link is a logical path between two cooperating tasks. These tasks must 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.

The task requesting to establish a logical link is called the source task. The other task is called the target task. This distinction applies only during the connection sequence. Once the logical link is established, the terms source and target have no significance, since both tasks have equal access to the logical link.

Tasks at either end of the link must specify a logical unit number (LUN) for the link. The LUN is the number each task assigns to the logical link so that the link can be associated by the tasks and the network. The tasks at both ends of the link do not have to use the same LUN for a link.

1.2.4 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, a source descriptor, access control information, and optionally, user-supplied data.

1.2.4.1 Destination Descriptor — When one task communicates with another task, the tasks are considered to be two objects communicating. There are two kinds of objects: named objects and numbered objects. The destination descriptor identifies the destination task either by task name or by object type number.

Named objects are installed user-defined tasks that are referred to by name when a connection is requested. The object type numeric identifier for user tasks referred to by name is 0.

Numbered objects are installed user-defined tasks and installed DECnet tasks that are referred to by object type number when a connection is requested. The object type numeric identifier for these tasks referred to by number ranges from 1 to 255. Numbers from 1 to 127 are reserved for DECnet tasks. Numbers from 128 to 255 are reserved for user tasks.

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

1.2.4.2 Source Descriptor — The source descriptor contains information supplied by the DECnet software on the source node. It contains the source node name and either the source task name if it is a named object or the source task object number if it is a numbered object. The target task can use this information to determine if it wants to establish communications or not.

1.2.4.3 Access Control Information — Access control information contains arguments that define your access rights at the remote node. Access control verification is performed according to the conventions of the target system. If the target node is equipped to do so, it verifies access control information before the connect request is passed to the target task. For information on access control verification, see the *DECnet-RSX Network Management Concepts and Procedures* manual.

1.2.4.4 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.

1.2.5 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 must issue 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 above four options.

1.2.6 Sending and Receiving Messages

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

To send a data message, a task issues a send (SND*x*) call. In the send call, specify the LUN assigned in the connect or accept call. Also specify 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. In the receive call, specify the LUN assigned in the connect or accept call. Also specify 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. In the case of overrun, the I/O status indicates this. To receive the next data message, another receive call is required.

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.

For more information on sending and receiving messages, see Sections 3.7.15 and 3.7.13.

1.2.7 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*) call 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 has 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 is returned with a specific error code in the I/O status word.

1.2.8 Checking Completion Status Information

Each macro or call can include an argument specifying the address of a 2-word status block. While this is an optional argument, you should include it. There is no other way to check the status of a call when it returns. Each active macro and call should have its own status block. The use of the same status block by concurrent I/O requests will produce unpredictable results.

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.

1.2.9 Terminating Activity on a Logical Link

Any task can terminate activity on a logical link at any time. To do so, you can issue a disconnect call or an 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, regardless of any messages 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.

In both disconnect and abort calls, you can specify the location and length of a user data message for the receiving task. The message can be up to 16 bytes long.

In the disconnect call, specify the logical unit number (LUN) assigned in the connect or accept call. When a disconnect call is issued, the software causes all pending transmits for the task issuing the disconnect call to complete before disconnecting the logical link. During this time, the task issuing the disconnect call continues to receive messages. When the last message is transmitted, 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.

In the abort call, specify the logical unit number (LUN) assigned in the connect or accept call. When an abort call is issued, the software immediately aborts all pending transmits and receives and disconnects the link. The LUN is freed and a task can use that LUN in subsequent connect or accept calls.

1.2.10 Closing a Network Connection

To close a task's network connection, issue a close (CLS*x*) call. The close call informs the software that the task no longer requires network services. This causes the software to purge 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 there is data in the task's network data queue when the close call is issued the following can occur:

- If the terminating task's network data queue contains any connect requests, the terminating task will receive them if it subsequently issues an open call within a short period of time.
- Any other type of data in the terminating task's network data queue is discarded (for example, interrupt, disconnect, and abort messages).

Table 1–1: DECnet Communication Calls Summary

Call	Function	Normal Action
ABT$ ABTNT	Abort a logical link	Abort is a nondata message transmitted over the logical link. It is delivered to the receiving task's network data queue by the DECnet software on the receiving task node.
ACC$ ACCNT	Accept a logical link request	Notification of the target task's acceptance of the logical link request is sent by the DECnet software on the target task node to the DECnet software on the source task node, which delivers it to the status block of the source task CON$ or CONNT call.
BACC	Build access control information area	No user data is transmitted over a logical link by this call. In a subsequent CONNT call, the contents of this area are delivered to the DECnet software on the target task node.
BFMT0 BFMT1	Build a format descriptor block	No user data is transmitted over a logical link by this call. In a subsequent CONNT call, the contents of this block are delivered to the DECnet software on the target task node.
CLS$ CLSNT	Close the network connection — end the task's network operations	No user data is transmitted over a logical link by this call. The close request is delivered to the DECnet software on the issuing task node.
CON$ CONNT	Request a logical link connection	A connect request is a high priority nondata message. The connect request and the connect block are sent over the temporary logical link to the DECnet software on the target node. If the target task is an active network task, the connect request is delivered to the target task's network data queue. The connect block is delivered to a mailbox as a result of a subsequent get network data call (GND$ or GNDNT).
CONB$$	Build a connect block	No user data is transmitted over a logical link by this call. In a subsequent CON$ call the contents of this block are delivered to the DECnet software on the target task node.
DSC$ DSCNT	Disconnect the logical link	Disconnect is a nondata message transmitted over the logical link. It is delivered to the receiving task's network data queue by the DECnet software on the receiving task node.
GLN$ GLNNT	Get local node data: node name and transmission segment size	No user data is transmitted over a logical link by this call. The requested data is delivered to locations specified by arguments of the call in the issuing task.

(continued on next page)

Table 1–1 (cont.): DECnet Communication Calls Summary

Call	Function	Normal Action
GND$ GNDNT	Get network data from task's network data queue	No user data is transmitted over a logical link by this call. The requested data is delivered to the location specified by arguments of the call in the issuing task.
OPN$ OPNNT	Open the network connection — create the task's network data queue	No user data is transmitted over a logical link by this call. The open request is delivered to the DECnet software on the issuing task node.
REC$ RECNT	Request to receive data over the logical link	No user data is transmitted over a logical link by this call. Notification that this call completed (that is, that the task that issued this call received a data message as a result of another task issuing a DECnet send call) is delivered to the status block of this call by the DECnet software on the issuing task node.
REJ$ REJNT	Reject a logical link request	Notification of the target task's rejection of the logical link request is sent over the temporary logical link by the DECnet software on the target task node to the DECnet software on the source task node, which delivers it to the status block of the source task CON$ or CONNT call.
SND$ SNDNT	Request to send a data message over the logical link	The data message is transmitted over the logical link to the DECnet software on the receiving task node for delivery to the area specified in a DECnet receive call issued by the receiving task. When the DECnet software has delivered the data to the receiving task, it sends a completion status message over the logical link to the DECnet software on the sending task node. The completion status message is delivered to the status block of the send call.
SPA$	Specify the location of a user-writen asynchronous system trap (AST) routine.	No user data is transmitted over a logical link by this call. When a nondata message is placed on the task's network data queue, control is transferred to the AST.
WAITNT	Wait for the completion of any other DECnet communications call.	No user data is transmitted over a logical link by this call. Suspends task execution until a previously issued call that had the wait option specified completes.
XMI$ XMINT	Request to send an interrupt message over the logical link.	Interrupt is a high priority nondata message transmitted over the logical link. It is delivered to the receiving task's network data queue by the DECnet software on the receiving task node.

1.2.11 Using the Wait Option

Many macros and calls allow you to include a W in the call name. When a W is included in a call (GNDW$ or GNDNTW), execution of the calling task is delayed until the indicated call request has completed. Execution of the calling task then proceeds at the instruction immediately following the call. This ensures that the process to be performed by the macro or call completes before the task continues. When the wait option is not used, the call is executed asynchronously.

NOTE

When using the wait option in a MACRO-11 call, you must assign an event flag. If the event flag is not specified, the call completes as a normal asynchronous call.

1.2.12 Using the AST and WAITNT Options

Where an asynchronous system trap (AST) has been specified in a MACRO-11 call, the AST is executed when the call completes. In FORTRAN, COBOL, and BASIC-PLUS-2, the WAITNT call is provided to determine when a call completes because there is no AST available.

1.2.13 Using the Flow Control Option

Network programs require buffer space for temporary message storage. For example, buffer space is needed to keep a copy of every message that it sends over the link until the receiver acknowledges receipt of the message. Network programs hold buffer space while waiting for inbound messages to be received.

DECnet provides flow control mechanisms to prevent the overflow of available buffer space. It forces synchronization between the sending and receiving tasks so that data is transmitted from a source task 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 the flow control mechanisms. The NOFLOW option can be beneficial when a higher level of network performance is desired. However, this option must be used with caution. Without flow control, a source task can send data regardless of the availability of a buffer at the receiving end. If the target task does not have adequate buffering to handle the incoming flow of data, some data segments will be discarded. Each time a data segment is discarded, the software must request retransmission of

the discarded segment, after a timeout. This significantly degrades network performance. If you choose the NOFLOW option, adequate buffering should be maintained 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 under the following conditions:

- Programs already have control mechanisms or acknowledgment-signaling mechanisms written into their User layer protocol.
- The flow of data is predictable, and the program can anticipate and handle the flow.

NOTE

If you are developing network programs, you are advised not to specify the NOFLOW option initially. Specify the NOFLOW option only when the communicating programs have been tested and the data flow between them is adequately synchronized.

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

1.3 DECnet–RSX Remote File Access Operations

Using DECnet–RSX remote file access facilities, you can write a FORTRAN, COBOL, or BASIC-PLUS-2 program to perform 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 required by the system. Unlike intertask communication, the DECnet software establishes the logical link, so much of the connection process is transparent. After completing file access operations, the logical link is disconnected.

A discussion of remote file access operations is provided in Section 3.8.

1.4 DECnet–RSX Task Control

DECnet–RSX task control allows you to write tasks in FORTRAN that perform the following activities:

- 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

A discussion of remote task control is provided in Section 3.9.

2
DECnet–RSX MACRO–11 Programming Facilities

DECnet–RSX provides a library of MACRO–11 macros that you can use to perform network intertask communication. DECnet–RSX MACRO–11 programming facilities describe those macros and explain how to use them.

2.1 RSX–11 Network Macro Formats

You can use the following formats to code your 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 created with a BUILD type macro and executes the function requested. An EXECUTE type macro allows you the option of overriding parameters specified by the BUILD type macro.
- **STACK type macro.** Creates a parameter block on the processor stack and executes the requested function.

Examples demonstrating the use of these three macro types are provided in Section 2.1.4.

2.1.1 BUILD Type Macros

The BUILD type macro is used at assembly time to create a parameter block that contains arguments describing the particular network function you have requested. Having a predefined parameter block of this sort is especially useful if you are going to

perform the same network operation a number of times. If you omit an optional parameter from this block, the macro allocates space for it anyway. This allows you to fill in the argument at a later time using an EXECUTE type macro. However, if you plan to use the EXECUTE type macro, you must include all trailing arguments in the BUILD type macro.

The format for issuing 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. A list of possible parameters for each macro is contained in the discussion of each macro. 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.

BUILD type macros can be executed 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 have created with the BUILD macro on the processor stack and then issues an EMT 377 for the Executive to execute the macro. A DIR$ macro can be written as follows:

DIR$ *adr,err*

where

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

NOTE

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 allows you to enter parameters that override parameters that were defined when you originally built the parameter block with a BUILD macro.

NOTE

All trailing arguments must be included in a BUILD type macro that is referenced by an EXECUTE type macro.

Once you have redefined or specified new parameters for the call, the EXECUTE macro automatically executes the function requested by the call. The format for issuing 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 that you specify to replace parameters that were previously defined for this call using a BUILD type macro. Each argument in this list must be a valid source operand for a MOV S, label+offset MACRO–11 instruction.

The value of a parameter can be overridden and assigned a null value by specifying a null value for that parameter. For example, if an AST is not required, the parameter should be specified 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 function requested. All required parameters must be specified when you issue this macro. Otherwise, the macro generates assembly errors.

The format for issuing a STACK type macro is:

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

where

xxx	is the name of a DECnet–11 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 describes particular features of this call. The arguments can be one or more legal source operands for MOV S,-(SP) MACRO–11 instructions. A list of possible parameters for each call is contained in the discussion of each 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.

label: xxx[W]$ *lun,efn,status,ast,*<*p*1,*p*2,...,*pn*>

```
;Create a parameter block
;for the call designated
;by xxx$ using a BUILD
;type macro.
```

xxx[W]$E *label*

```
;Issue an EXECUTE type macro
;referencing the parameter
;block created for label.
```

xxx[W]$E *label*,,,,,<*p*1,*p*2>

```
;Issue an EXECUTE type macro
;and override the parameter
;list arguments p1 and p2.
```

xxx[W]$S #*lun*,#*efn*,#*status*,#*ast*,<#*p*1,#*p*2>

```
;Issue a STACK type macro, create
;a parameter block on the stack,
;and execute the call.
```

2.2 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 — represent actual characters that you must enter as shown.

lowercase italic — indicates variables whose value you must specify.

square brackets [ ] — enclose optional data. You must specify any argument not enclosed by brackets. 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 several arguments of which you must select just one. Do not type the braces when you code the macro.

Example:

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

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

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 where shown as part of the macro format. Even if you omit an argument, you must 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 assumed to be 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.

2.3 Intertask Communication Macros

This section contains descriptions and usage guidelines specific to the intertask communication calls listed in Table 2–1. Before turning to these calls, you should read the preceding material in this chapter. If you are not familiar with intertask communication concepts, you should also read Chapter 1 carefully before you attempt to code any of these calls.

Table 2–1: Intertask Communication Macro Summary

Macro	Function
ABT$	Abort a logical link
ACC$	Accept a logical link connect request
CLS$	End a task's network operations
CON$	Request a logical link connection
CONB$$	Build connect 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

2.3.1 Common Argument Definitions

Arguments that are commonly used in intertask communication macros are defined below to avoid needless repetition throughout the macro descriptions.

- *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. If desired, you can override any arguments defined in that BUILD macro 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 (see individual macro descriptions for possible codes)

  Byte 1 = 0

  Word 1: 0

- *out,outlen*

  defines optional user data you wish to send with certain macros. These are paired optional arguments; use both when specified, 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 you wish to send.

2.3.2 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 definition in Section 2.3.1.

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 that is set when ABT$ completes.

* *status*

specifies completion status information on return from ABT$. See definition in Section 2.3.1.

ast

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

out,outlen

defines optional user data you wish to send. See definition in Section 2.3.1.

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.

2.3.3 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 definition in Section 2.3.1.

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 that is set when ACC$ completes.

* *status*

specifies completion status information on return from ACC$. See definition in Section 2.3.1.

ast

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

mail

is the address of the connect block sent by the source task and retrieved by GND$. This address is the same one that is specified for *mail* in GND$ (see Section 2.3.9). Connect block information (see Table 2–5) is needed to establish the connection.

mailen

is the length in decimal bytes of the connect block. If omitted, the value N.CBL (98.) is used (see Table 2–5).

out,outlen

defines optional user data you wish to send. See definition in Section 2.3.1.

Flag:

NOFLOW

disables flow control for incoming messages (that is, messages destined for the task that issued ACC$). If NOFLOW is omitted, flow control is established for incoming messages. Either end of the link can be set independently to flow or noflow control. Use the NOFLOW option with caution (see Section 1.2.13).

Error/Completion Codes:

IS.SUC The macro completed successfully.

IE.ABO The task that requested the connection has aborted or has 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.

2.3.4 CLS$ – End Task Network Operations

Use:

Issue CLS$ from either task to end that task's network activity, abort all its logical links, and free all its network LUNs. If there is data in the task's network data queue, the following results can occur:

- If the queue contains any pending connect requests that arrived while the task was active, the calling task is rescheduled (that is, the task will receive these connect requests whenever it is restarted). There is a limit of one retry and a timeout period of approximately 15 seconds.
- If any connect requests arrived before the task was active, they are rejected.
- If the queue contains an interrupt message, user disconnect, user abort, or network abort, this data is discarded.

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 definition in Section 2.3.1.

* *efn*

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

* *status*

specifies completion status information on return from CLS$. See definition in Section 2.3.1.

ast

is the address of an optional user-written AST routine to be executed 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.

2.3.5 CON$ – Request Logical Link Connection

Use:

Issue CON$ from the source task to request a logical link with the target task. Before you issue CON$, you must build a connect block using the CONB$$ macro (see Section 2.3.6). This connect block is passed to the target node when you issue CON$.

When a remote system receives a connect request, it checks to see if the target task is currently installed and inactive. If it is, it automatically loads and activates the task. The target task must issue a GND$ (see Section 2.3.9) to retrieve the connect block information, which it evaluates to determine whether to accept (ACC$, see Section 2.3.3) or reject (REJ$, see Section 2.3.12) the connect request.

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 definition in Section 2.3.1.

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 that is 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 list below)
Byte 1 = 0

Word 1: Byte 0 = Content depends on error completion code in word 0, byte 0 (see list below)
Byte 1 = 0

Listed below are possible error/completion codes you can receive in word 0, byte 0, plus 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 received)
IS.DAO Connection accepted with data overrun	Received byte count (0 if no data received)
IE.DAO Connection rejected by user with data overrun	Received byte count (0 if no data received)
IE.URJ Connection rejected by user	Received byte count (0 if no data received)
IE.NRJ Connection rejected by DECnet	Reason for rejection (see Appendix A)
All other cases	0

ast

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

conbl

is the address of the connect block built using CONB$$ (see Section 2.3.6). This block must start on an even byte (word) boundary.

conblen

is the length of the connect block in decimal bytes. If omitted, the argument N.RQL (72.) is used (see Table 2–2).

out,outlen

defines optional user data you wish to send. See definition in Section 2.3.1.

in,inlen

defines 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. If NOFLOW is omitted, this end of the link is established as flow controlled. Either end of the link can be set independently to flow or noflow control. Use the NOFLOW option with caution (see Section 1.2.13).

Error/Completion Codes:

IS.SUC The macro completed successfully.

IS.DAO The macro completed successfully; the connection has been accepted, but some optional user data was lost (the data sent from the target task when it accepted your connect request).

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 was lost (the data sent from the target task when it rejected your connect request).

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 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.

2.3.6 CONB$$ – Build Connect Block

Use:

Issue CONB$$ from the source task to build a 72.-byte connect block to be passed to the target task when you issue CON$ (see Section 2.3.5). The connect block contains the node name, destination descriptor, and access control information required by the target task to determine whether to accept (ACC$) or reject (REJ$) the connect request. Figure 2–1 illustrates a sample connect block.

Access control information comprises arguments that define your access rights at the remote node or process. Access control verification is performed according to the conventions of the target system. If the target node is equipped to do so, it verifies access control information before the connect request (CON$) is passed to the target task. (For more information on access control verification, see the *DECnet–RSX Network Management Concepts and Procedures* manual.

NOTE

If you have already included the correct access control information with an alias node name, you need not include it in the connect block. For more information on using aliases, refer to the *DECnet–RSX Guide to User Utilities* or to the *DECnet–RSX Network Management Concepts and Procedures* manual.

Format:

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

Arguments:

NOTE

If you must specify non-ASCII data for any of the following arguments that require ASCII data, leave the argument field blank and create the non-ASCII field dynamically. You can create the entire connect block dynamically by reserving a block of storage equal to the length of N.RQL (72. bytes, as shown in Table 2–2). If you do not include the arguments in the CONB$$ macro call, you must define the fields before the macro is executed. All fields can be created or modified during task execution by using the symbolic offsets shown in Table 2–2. Use the CRBDF$ call to define these symbolic offsets.

node

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

The *obj*, *fmt*, and *descrip* arguments, described below, are collectively referred to as the destination descriptor. The destination descriptor describes formatting characteristics required by the target task. You must specify this information in order to access the task.

obj

is the object type of the task to which the connect request is directed. Object types group DECnet tasks according to the functions they perform; they are identified throughout the network by object type codes (see list of codes in Appendix B). Your code must be in the range of 0 through 255 (decimal).

NOTE

If you are a privileged user, you can define your own object types; refer to the *DECnet–RSX Network Management Concepts and Procedures* manual for instructions.

fmt

is the descriptor format type. If you specified *obj* as 0, use 1 for *fmt*; otherwise, use 0.

descrip

is the target task name (1 to 16. ASCII characters). Omit this argument if you specified a nonzero object type.

The *rqid*, *pass*, and *accno* arguments specify access control information that determines your access at the remote node or process. If you have already provided this information in an alias node name for the target node, you can omit these arguments.

rqid

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

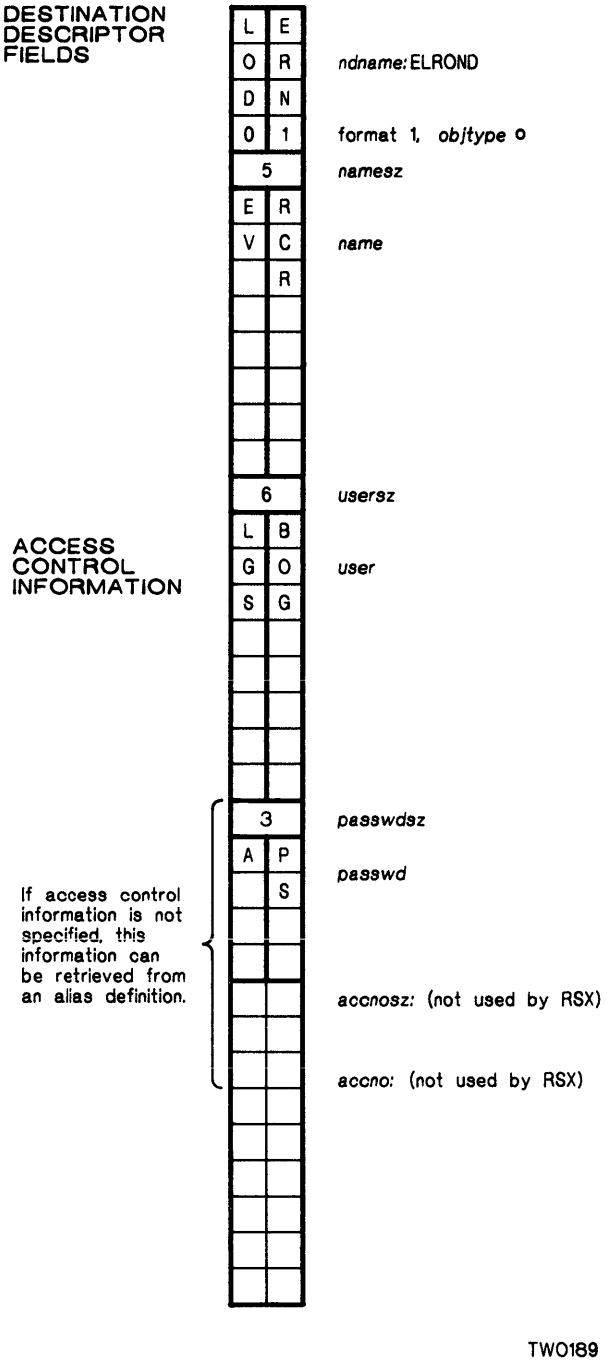


Figure 2–1: Sample Connect Block

pass

is a 1 to 8.-byte password. If you are entering an ASCII password (as opposed to binary), precede each character of the password with an apostrophe (') and separate the characters with commas. For example, the password PAS should be entered as 'P,'A,'S.

accno

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

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

Symbolic Offset	Length (decimal 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 (that is, word aligned).

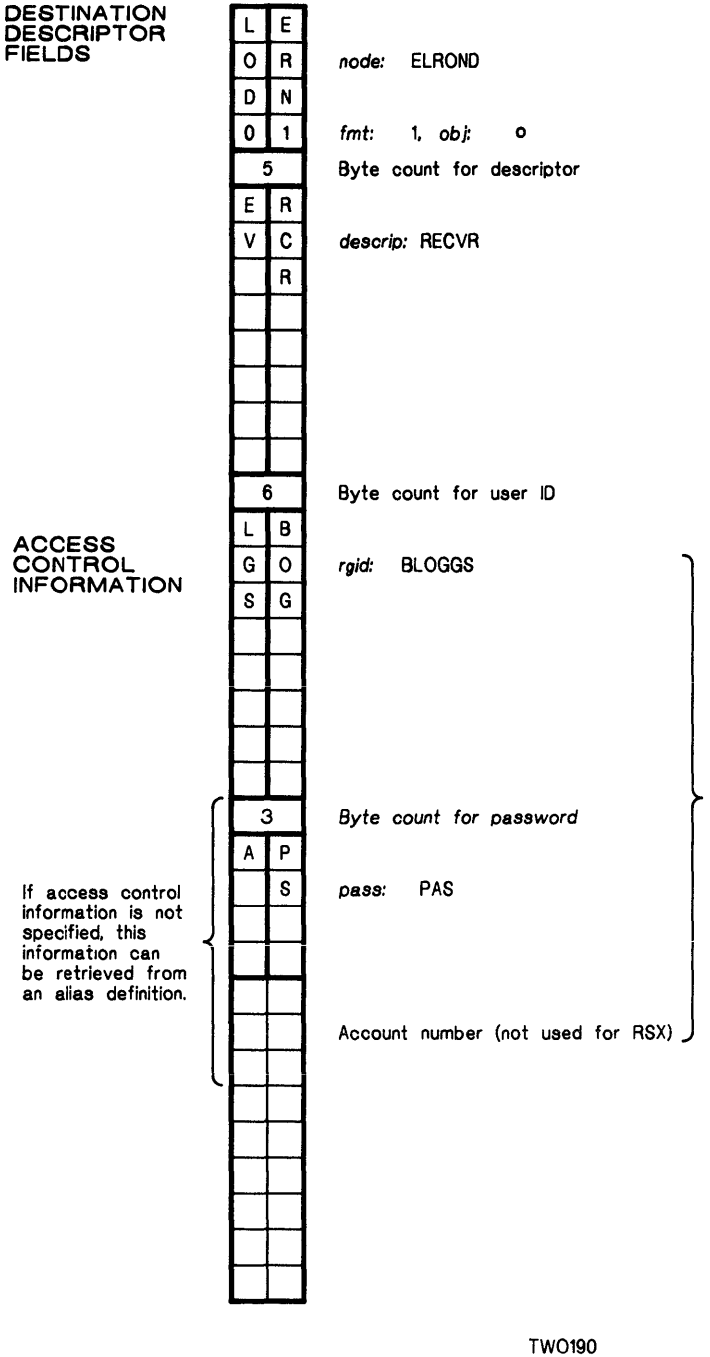


Figure 2–2: Sample Connect Block Built by CONB$$

Figure 2–2 illustrates a connect block that was built as a result of executing the following call:

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

This connect block will be directed to a task named RECVR on remote RSX node ELROND. The object type is 0 (named object) and the descriptor format type is 1. The user ID is BLOGGS and the password is PAS; no account number is required for RSX target systems.

2.3.7 DSC$ – Disconnect a Logical Link

Use:

Issue DSC$ from either task to disconnect the logical link and free the logical unit number. Unlike ABT$ (see Section 2.3.2), DSC$ causes all pending transmits to complete before the link is disconnected. While these transmits are completing, the task continues to receive messages. When the last transmit has completed, all pending receives are aborted and IE.ABO is returned in the I/O status block for each one. When you issue 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 definition in Section 2.3.1.

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 that is set when DSC$ completes.

* *status*

specifies completion status information on return from DSC$. See definition in Section 2.3.1.

ast

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

out,outlen

defines optional user data you wish to send. See definition in Section 2.3.1.

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.

2.3.8 GLN$ – Get Local Node Information

Use:

Issue GLN$ from either task to have the following local node information placed in the specified buffer:

- Local node name

  You may wish to supply the local node name within a task that is to connect to the local node. For example, if you have a task that runs on several nodes and sends data to the local node, your task must supply the local node name to the connect block before it can establish a connection with that node. You also may wish to issue a GLN$ within any task that is to display the local node name.

- Default NSP segment size (that is, the size that NSP uses to segment data transmitted on a logical link)

  When you know the default NSP segment size, you can use transmit buffers (large data buffers) most efficiently by adjusting the length of the message blocks to be transmitted.

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 definition in Section 2.3.1.

* *efn*

specifies an optional event flag number that is 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 depends 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 be executed 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. On return from GLN$, the data is stored as follows:

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

buflen

is the length of the buffer (6 or 8. bytes) to contain the received data. If you specify 6 bytes, only the local node name will be returned. If you specify 8. bytes, both the node name and the default NSP segment size will be returned.

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.

2.3.9 GND$ – Get Network Data

Use:

Issue GND$ from either task to get data from that task's network data queue and store it in the specified mail buffer (see *mail,mlen*). If GND$ completes successfully, word 0, byte 1 of the status block identifies which of the following unsolicited message types has been stored:

- 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 (see Section 2.3.14) to find out how many network data items are in the network data queue. If you issue GND$ when the queue is empty, GND$ completes with an error (IE.NDA) even if the GND[W] form is used.

Formats:

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

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

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

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

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

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

Table 2–3: Contents of Status Block after GND$

If GND$ completes successfully and NT.LON is not specified:

Status Word 0 Byte 0	Byte 1	Status Word 1 Byte 0	Byte 1
IS.SUC or IS.DAO or IE.DAO	NT.CON Connect request	Number of bytes in connect block (see Table 2–4).	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 and NT.LON is specified:

Status Word 0 Byte 0	Byte 1	Status Word 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

(continued on next page)

Table 2–3 (cont.): Contents of Status Block 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 for the target node, it evaluates access control data in the connect request before it is allowed to pass to the target task's network data queue. For more information on access control, refer to the *DECnet–RSX Network Management Concepts and Procedures* manual.
2. Either 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.
3. Either 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.

Arguments:

label

specifies the location of the argument block. See definition in Section 2.3.1.

lun

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

* *efn*

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

* *status*

is the address of an optional 2-word status block that contains completion status information on return from GND$. The content of the status block is summarized in Table 2–3.

ast

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

mail,mlen

defines the task mail buffer to receive the network data or connect block on return from GND$. (Connect block contents are itemized in Table 2-5.) These arguments must be specified if the NT.TYP and NT.LON flags are omitted.

* *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 (98. to 116.).

mask

specifies data type to be selected from the network data queue. Normally, GND$ returns items from the network data queue on a first-in, first-out basis. However, you can use *mask* to select the first item on the queue that matches the message type and/or LUN that you choose. Enter one of the combinations given in Table 2-4 for the *mask* argument:

Table 2–4: Mask Argument Options

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 message)	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, if you want to select from the network data queue the first disconnect message (NT.DSC) on LUN 3, you would code the *mask* argument as follows: 3*256.+NT.DSC.

If you specify 0 in byte 1, the first message of the type specified in byte 0 will be returned, regardless of the LUN.

Flags:

NT.TYP

indicates that a specific message type and/or LUN has been requested in a *mask* argument (see above). NT.TYP should always be used when *mask* is specified with *mail* and *mlen*.

NOTE

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

NT.LON

specifies dynamic assignment of mail buffer space. When you specify NT.LON, the message type of the first message in the network data queue is returned in word 0, byte 1 of the status block, and the message length is returned in word 1, byte 0. The message is not removed from the queue or placed in the mail buffer. If you specify NT.LON, you must not use *mail, mlen*, and NT.TYP.

NOTE

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

Figure 2–3 shows sample connect block information retrieved from the network data queue by a GND$ macro. The connect block shown is the one generated by the CONB$$ macro example given in Section 2.3.6.

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.

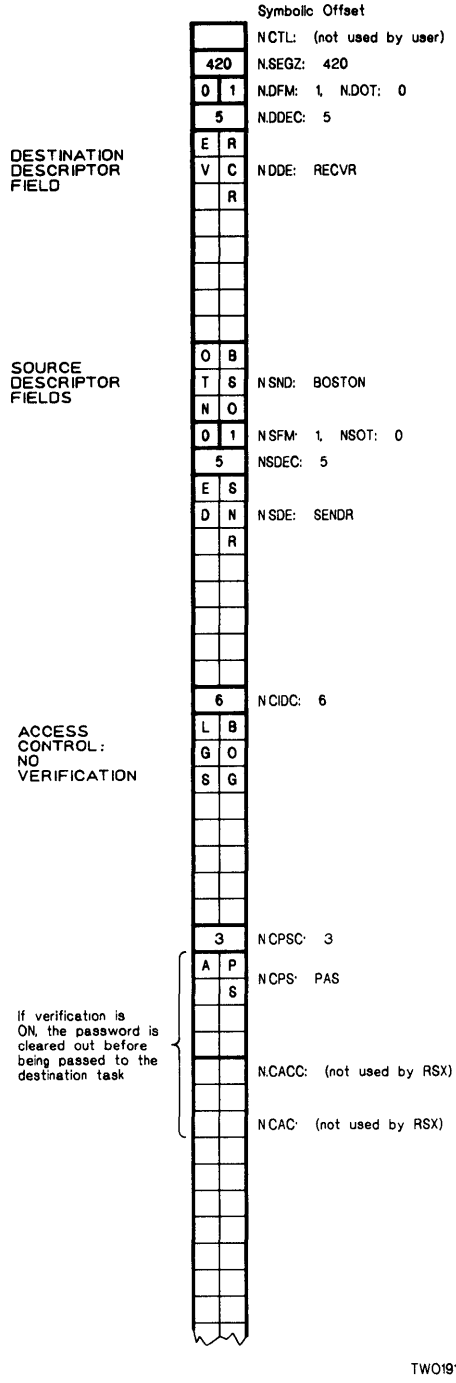


Figure 2–3: Sample Connect Block Received by GND$

Table 2–5: Connect Block Received in the Mail Buffer after GND$

Symbolic Offset	Length (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
		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 (must be either format 0 or format 1)

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

(continued on next page)

Table 2–5 (cont.): Connect Block Received in the Mail Buffer after GND$

Symbolic Offset	Length (decimal bytes)	Contents
		SOURCE DESCRIPTOR
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 task name length (equal to or less than 16. bytes)
		N.SDE* 16. Source task name
		ACCESS CONTROL INFORMATION (46.-byte total)
		If no verification 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)

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

(continued on next page)

Table 2–5 (cont.): Connect Block Received in the Mail Buffer after GND$

Symbolic Offset	Length (decimal bytes)	Contents
		ACCESS CONTROL INFORMATION
		N.CAC* 16. Account number
		If verification 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
		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)

N.CBL = 98. (not including optional data)

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

2.3.10 OPN$ – Access the Network

Use:

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

Formats:

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

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

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

Arguments:

label

specifies the location of the argument block. See definition in Section 2.3.1.

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. Use this LUN in any subsequent GND$, SPA$, GLN$, REJ$, or CLS$ macro.

* *efn*

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

* *status*

specifies completion status information on return from OPN$. See definition in Section 2.3.1.

ast

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

links

specifies the maximum number of logical links that can be active simultaneously 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 (which is also the default) sets no limit as long as network resources are available.

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 establish outgoing links by using CON$.

lrp

specifies the link recovery period — that is, the number of minutes that can elapse from the time a physical link fails until the associated logical link is aborted by the network. The *lrp* must be in the range of 0 through 32767(decimal).

When specifying an *lrp* value, remember that unless your task has been built checkpointable, it will be 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.

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.

2.3.11 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 definition in Section 2.3.1.

lun

specifies the logical link over which data is to be received. 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 that is set when REC$ completes.

* *status*

specifies completion status information on return from REC$. See definition in Section 2.3.1, and note this exception:

Word 1: Contains number of bytes received.

ast

is the address of an optional user-written AST routine to be executed 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.

2.3.12 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 definition in Section 2.3.1.

* *efn*

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

* *status*

specifies completion status information on return from REJ$. See definition in Section 2.3.1.

ast

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

mail

is the address of the connect block sent by the source task and retrieved by GND$. This address is the same one that is specified for *mail* in GND$ (see Section 2.3.9). Connect block information (see Table 2–5) is needed to reject the connection.

mailen

is the length in decimal bytes of the connect block. If omitted, the value N.CBL (98.) is used (see Table 2–5).

out,outlen

defines optional user data you wish to send. See definition in Section 2.3.1.

Error/Completion Codes:

IS.SUC The macro completed successfully.

IE.ABO The task that requested the connection has aborted or has 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.

2.3.13 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 definition in Section 2.3.1.

lun

identifies the logical link over which the data is to be sent. 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 that is set when SND$ completes.

* *status*

specifies completion status information on return from SND$. See definition in Section 2.3.1, and note this exception:

Word 1: Contains number of bytes sent.

ast

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

buf

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

buflen

is the length in bytes (8128. maximum) of the data you wish 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.

2.3.14 SPA$ – Specify User AST Routine

Use:

Issue SPA$ from either task to specify a user-written AST routine to be executed whenever network data arrives in the network data queue. No AST routine will be executed for data items that arrive in the queue before SPA$ is issued. However, a count of all data items in the queue (including premacro entries) is returned in word 1 of the SPA$ status block each time SPA$ is issued.

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 definition in Section 2.3.1.

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 that is set when SPA$ completes.

* *status*

specifies completion status information on return from SPA$. See definition in Section 2.3.1, and 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 be executed after SPA$ completes (see SPA$ programming note below).

addr

is the address of a user-written AST routine to be executed whenever data arrives in the network data queue. (If this argument is omitted, no AST routine is executed.) The specified AST routine can be changed during execution of the task by specifying a different starting address; it can be eliminated by zeroing the starting address.

NOTE

When this AST executes, no extra information is pushed onto the stack (as it is for a normal completion AST). 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 coding example shows how an application can process all network data at the AST level by using the SPA$ completion AST (specified by *ast* in the SPA$ macro) 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 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
```

2.3.15 XMI$ – Send Interrupt Message

Use:

Issue XMI$ from either task to send an interrupt message over an established logical link. The message you send is placed on the target task's network data queue and must be retrieved with a GND$ (see Section 2.3.9) 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 definition in Section 2.3.1.

lun

specifies the logical link over which the interrupt message is to be sent. 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 that is set when XMI$ completes.

* *status*

specifies completion status information on return from XMI$. See definition in Section 2.3.1, and note this exception:

Word 1: Contains number of bytes sent in message.

ast

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

int

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

intlen

is the length in decimal bytes of the message you wish to send.

Error/Completion Codes:

IS.SUC The interrupt message has been transmitted successfully. (However, this code does not ensure that the message has been retrieved by a GND$; see Section 2.3.9).

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.

2.3.16 MACRO–11 Intertask Communication Programming Example (Transmit)

The program SEN10 transmits 10 data messages to the cooperating program REC10. After SEN10 transmits and REC10 receives 10 data messages, both programs disconnect from the network.

NOTE

This programming example is also included in your tape or disk kit.

```
        .TITLE  SEN10
;*******************************************************************************
;*                                                                             *
;*  THIS EXAMPLE WILL :                                                        *
;*      SEND 10 DATA MESSAGES WITH THE FORMAT 'THIS IS MESSAGE N'              *
;*      ACCEPT A SHORT MESSAGE FROM THE INITIATING TERMINAL                    *
;*      AND SEND THIS MESSAGE OUT AS AN 'INTERRUPT MESSAGE'.                   *
;*                                                                             *
;*     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$$,ALUN$C,QIOW$C
        .MCALL  EXIT$S,MRKT$C,WTSE$C,CLEF$C,SETF$C,QIO$C
        .MCALL  DSCW$S,XMIW$S,ASTX$S
;
;  DATA AREA
;
MESN:   .ASCII  /THIS IS MESSAGE /      ; MESSAGE TO BE TRANSMITTED
NUM:    .ASCII  /0/                     ; MESSAGE NUMBER
NN=.-MESN
PRMPT:  .ASCII  /MSG:/                  ; PROMPT FOR INTERRUPT MESSAGE
        .EVEN
IOSTN:  .BLKW   2                       ; COMPLETION STATUS FOR NETWORK
BUFF:   .BLKB   16.                     ; INTERRUPT MESSAGE BUFFER
IOSTB:  .BLKW   2                       ; COMPLETION STATUS FOR BUFFER
CNT:    .WORD   0                       ; NUM OF CHAR IN INTERRUPT MESS
ERRCNT: .WORD   0                       ; ERROR COUNT
IOSB:   .BLKW   1                       ; I/O STATUS
;
        .EVEN
CONBL:  CONB$$  ELROND,0,1,<REC10>      ; CONNECT REQUEST BLOCK
;
;  CODE
;
        .EVEN
```

(continued on next page)

```
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$C  5                       ; MAKE SURE TERMINAL MESSAGE
                                        ; HAS BEEN ENTERED
                                        ; BEFORE EXITING
;
        DSCW$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$C  5                       ; SET EVENT FLAGE TO INDICATE
                                        ; INTERRUPT MESSAGE SENT
        ASTX$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
```

2.3.17 MACRO–11 Intertask Communication Programming Example (Receive)

Each time REC10 receives a message, from the cooperating program SEN10, it displays THIS IS MESSAGE *n* on the console device (CO:). This is followed by the actual message. The message is sent to REC10 as an interrupt message.

NOTE

This programming example is also included in your tape or disk kit.

```
        .TITLE  REC10
;*****************************************************************************
;*                                                                           *
;*  THIS EXAMPLE WILL:                                                       *
;*       ACCEPT SHORT MESSAGES FROM THE SENDER TASK "SND10"                  *
;*       PRINT THE MESSAGES ON THE CONSOLE DEVICE (CO:)                      *
;*       DISCONNECT AND EXIT 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
;
```

(continued on next page)

```
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
                                        ; HAS 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 \DDR
        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
```

(continued on next page)

```
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$C  10.                     ; SET EVENT FLAG TO INDICATE
                                        ; CONNECT HAS OCCURRED
        BR      NEXT
OTHER:  CMPB    #NT.DSC,IOST1+1         ; CHECK IF DISCONNECT REQUEST
        BNE     OTHR2
        MOV     #1,FLAG                 ; SET DISCONNECT FLAG
        BR      NEXT                    ; GO BACK TO MAIN ROUTINE
;
;
OTHR2:  CMPB    #NT.INT,IOST1+1         ; CHECK IF INTERRUPT MESSAGE
        BEQ     OKC
        JMP     ERR6                    ; NOT A EXPECTED COMMAND
OKC:    MOVB    IOST1+2,CNTB            ; OBTAIN CHARACTER COUNT
        QIOW$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

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

DECnet–RSX provides three types of network subroutines:

- Intertask communication calls
- Remote file access calls
- Task control calls (FORTRAN only)

Calls to perform these subroutines are listed 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 do not have to assign logical unit numbers (LUNs) for calls to the network (NS:) at task build time or in your program. The LUNs are assigned to the network at run time by the OPNNT[W], CONNT[W], and ACCNT[W] calls. If you do assign a LUN to the network at task build time or in your program, this 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 they are 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 the task.

OPNNTW Performs the same functions as OPNNT, except that further task execution is suspended until this call completes.

After opening the task to the network, you can establish a logical link by issuing calls as 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 functions as CLSNT, except that further task execution is suspended until this call completes.

3.3 Examining I/O Status Blocks

All calls listed in this chapter allow you to specify the address of a status block. This address will contain completion status information when a call completes.

NOTE

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

Status blocks are either 1- or 2-element integer arrays/strings. One-element arrays/strings are used for the BACC, BFMT0, and BFMT1 calls. In these 1-element arrays/strings, a return of –1 indicates the arguments you supplied for the call are valid; a 0 code indicates the arguments are invalid.

Calls other than BACC, BFMT0, and BFMT1 use 2-element arrays/strings. In a 2-element array/string, the first status word contains an error/completion code for the call. The error/completion code tells you one of the following:

- A positive value indicates the call executed successfully.
- A negative value indicates the call failed to 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. A complete list of error/completion codes is provided in Appendix E for intertask calls and task control calls.

The contents of the second status word can differ according to the call you issue. Therefore, each call 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. By default, the event flags used are 17 (.TREF) and 18 (.RCEF). You have the option of overriding 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 Obtaining Access Control Information

Access control information is generally required by the target system to prohibit unauthorized access to its resources. The specific information required by a target node should be contained in the target system's user documentation. DECnet–RSX nodes require you to enter user identification and a password. The following paragraphs describe how to supply access control information using intertask communication, remote file access, and alias node names.

- **Intertask communication calls.** If required by the target node, access control information is supplied by the source task user in the BACC call.
- **Remote file access calls.** If required by the target node, access control information is supplied by the source task user in the *ident* argument for individual calls.
- **Alias node names.** You can define an alias node name (a user-assigned logical name for a network node) and include access control information with the alias. When you exercise this option as a source task user, you do not have to specify access control fields in your program; the access control information supplied with the alias will be used automatically. More information on creating and using alias node names is provided in the *DECnet–RSX Network Management Concepts and Procedures* manual.

3.6 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 where completion status information is stored when the call completes.

UPPERCASE — represent actual characters that you must enter as shown.

lowercase italic — indicates variables whose value you must specify.

commas, periods parentheses ( ) — must be typed where shown as part of the call format. Even if you omit an argument, you must 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 )
```

Status, usersz, user, accnosz, and *accno* have been omitted. Commas delineate the fields for the first three missing arguments, but are not necessary 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 43.-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 do not wish to specify, but that has trailing arguments you wish to include.

COBOL Example:

Basic call format:

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

In this call there are three categories of optional data:

- *status*

  is optional, but it cannot be omitted because it is followed by a required argument (*tgtblk*). You can enter 0 for *status* if you do not want to have status information returned on the call.

- *outsize,outmessage*

  are paired optional arguments you can omit only if you also omit the trailing arguments, *insize* and *inmessage*. If you do not want to specify *outsize* and *outmessage*, but do want to include the arguments that follow, you can enter null arguments (0) for *outsize* and *outmessage*.

- *insize,inmessage*

  are paired optional arguments 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.

3.7 Intertask Communication

This section contains descriptions and usage guidelines specific to the intertask communication calls listed alphabetically in Table 3–1.

Before turning to these calls, you should read the preceding material in this chapter. If you are not familiar with network intertask communication concepts, you should also read Chapter 1 carefully before you attempt to code any of these calls.

Table 3–1: Intertask Communication Call Summary

Call	Function
ABTNT	Abort a logical link
ACCNT	Accept a logical link connect request
BACC	Build access control information area
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*)

3.7.1 Common Argument Definitions

Arguments commonly used in intertask communication calls are defined on the following pages to avoid needless repetition throughout the call descriptions. Argument definitions here are divided into four categories: a general category containing definitions common to all languages and three individual language categories for arguments with language-specific definitions.

GENERAL

- *outsize,outmessage*

  defines optional user data you want to send with certain calls. These are paired optional arguments; use both when specified, 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; if you want to include *insize,inmessage*, but do not want to specify *outsize,outmessage*, you can enter a null value (0) for both *outsize* and *outmessage*. (See the example under the discussion of square brackets [] in Section 3.6.)

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 you want 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 optional 2-element single-precision integer array will contain the following values when the call completes:

  status(1) returns an error/completion code (see individual call descriptions for possible codes).

  status(2) returns a directive error code if *status*(1) returns a value of -40; otherwise, *status*(2) contains 0.

- *tgtblk*

  specifies a 72.-element byte array where the access control information area and destination descriptor are built by the BACC and BFMT0 or BFMT1 calls. This array is passed to the target task in a CONNT call. The array must start on an even byte (word) boundary.

COBOL

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

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

  status(1) returns an error/completion code (see individual call descriptions for 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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.
- *tgtblk*

  specifies a 72.-element numeric data item where the access control information area and destination descriptor are built by the BACC and BFMT0 or BFMT1 calls. This is passed to the target task in a CONNT call.

BASIC-PLUS-2

- *status*%()

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

  status%(0) returns an error/completion code (see individual call descriptions for 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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.
- *tgtblk*$

  specifies a 72.-element character string where the access control information area and destination descriptor are built by the BACC and BFMT0 or BFMT1 calls. This string is passed to the target task in a CONNT call. To allocate space for *tgtblk*$, use the STRING function:

```
tgtblk$=STRING$(72,0)
```

3.7.2 ABTNT – Abort a 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 LUN assigned 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. 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 ABTNT. See definition for your language in Section 3.7.1.

outsize,*outmessage*

defines optional user data you want to send. See definition in Section 3.7.1.

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.*

3.7.3 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 definition for your language in Section 3.7.1.

mailbuf

specifies the 1- to *n*-element array/string containing the connect block needed to establish the connection. 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 3.7.11).

outsize,outmessage

defines optional user data you want to send. See definition in Section 3.7.1.

Error/Completion Codes:

1 The call completed successfully.

-1 System resources needed for the logical link are not available.

-3 The task that requested the connection has aborted or has 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.*

3.7.4 BACC – Build Access Control Information Area

Use:

Call BACC from the source task to build the access control information area for the connect block that will be passed to the target task when you call CONNT (see Section 3.7.8). Access control information comprises arguments that define your access rights at the remote node or process. Access control verification is performed according to the conventions of the target system. If the target node is equipped to do so, it verifies access control information before the CONNT call is passed to the target task. (For more information on access control verification, see the *DECnet–RSX Network Management Concepts and Procedures* manual and Sections 1.2.4.3 and 3.5 of this manual.)

NOTE

If you have already included the correct access control information with an alias node name, you need not call BACC. For more information on using aliases, refer to the *DECnet–RSX Guide to User Utilities* or to the *DECnet–RSX Network Management Concepts and Procedures* manual.

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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.

* *tgtblk*

specifies an array/string in which the access control information area is to be built. See definition for your language in Section 3.7.1.

usersz,user

specifies the user ID. These are paired optional arguments; if you do not use both, you must omit both (in FORTRAN) or enter 0 for both (in COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets [ ]) in Section 3.6.

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

specifies the password that determines your access at the remote node. These are paired optional arguments; if you do not use both, you must omit both (in FORTRAN) or enter 0 for both (in COBOL and BASIC). For information on omitting arguments in COBOL and BASIC, refer to the discussion of optional arguments (square brackets [ ]) in Section 3.6.

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

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

accnosz specifies the account number length in bytes/characters (not used 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 (see Sections 3.7.5 and 3.7.6, respectively)
	Access Control
2.	User ID length (equal to or less than 16. bytes/characters)
16.	User ID
2.	Password length (equal to or less than 8. bytes/characters)
8.	Password
2.	Account number length (equal to or less than 16. bytes/characters)
16.	Account number

3.7.5 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 that will be passed to the target task when you call CONNT (see Section 3.7.8). A format 0 descriptor is used to connect to a target task that requires specification of an object type only. Object types group DECnet programs according to the functions they perform; they 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 providing the same function on another DECnet system, regardless of its name, would also be referred to using object type code 016 (decimal).

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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.

* *tgtblk*

specifies an array/string in which the destination descriptor is to be built. See definition for your language in Section 3.7.1.

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 node to which the connect request is directed.

objtype

specifies the object type of the task to which the connect request is directed. This integer variable or constant must be in the range 1 through 255 (decimal). See Appendix B for a list of object types.

NOTE

If you are a privileged user, you can define your own object types; refer to the *DECnet-RSX Network Management Concepts and Procedures* manual for instructions.

Connect Block Offsets

Length (in decimal bytes/characters)	Destination Descriptor
6.	Destination node name with trailing blanks
1.	Descriptor format type, which is 0 for BFMT0
1.	Destination object type (1 to 255.)
	Descriptor Field for Format 0
18.	Not used
	Access Control
46.	Built by BACC call (see Section 3.7.4)

3.7.6 BFMT1 – Build a Format 1 Destination Descriptor

Use:

Call BFMT1 from the source task to build a format 1 destination descriptor for the connect block that will be passed to the target task when you call CONNT (see Section 3.7.8). A format 1 descriptor is used to connect to a target task that requires specification of a task name only.

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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.

* *tgtblk*

specifies an array/string in which the destination descriptor is to be built. See definition for your language in Section 3.7.1.

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 containing the name of the node to which the connect request is directed.

objtype

specifies the object type to which the connect request is directed. 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 16.-element array/string containing the name of the program to which you wish to connect.

Connect Block Offsets

Length (in decimal bytes/characters)	Destination Descriptor
6.	Destination node name with trailing blanks
1.	Descriptor format type, which is 1 for BFMT1
1.	Destination object type, which is 0 for BFMT1
	Descriptor Fields for Format 1
2.	Destination program name length (equal to or less than 16. bytes/characters)
16.	Destination program name
	Access Control
46.	Built by BACC call (see Section 3.7.4)

Sample coding for BFMT1 calls is given in the following examples. Each language-specific example shows 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(72)

DATA NDNAM/'E','L','R','O','N','D'/
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 "ELROND".
   03 TSKNAM    PIC X(6) VALUE "RECVR".
   03 STAT      PIC X999 USAGE COMP.
   03 CONBLK    PIC X(72).
   03 NLENG     PIC 9 USAGE COMP.
   03 TLENG     PIC 9 USAGE COMP.
   03 DUMMY     PIC X (2).
                        .
                        .
                        .
PROCEDURE DIVISION
                        .
                        .
                        .
```

(continued on next page)

```
*********************************
* 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(72%,0%)

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

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

3.7.7 CLSNT – End Task Network Operations

Use:

Call CLSNT from either task to end that task's network activity, abort all its logical links, and free all its network LUNs. If there is data in the task's network data queue, the following results can occur:

- If the queue contains any pending connect requests that arrived while the task was active, the calling task is rescheduled (that is, the task will receive these connect requests whenever it is restarted). There is a limit of one retry.

  If any connect requests arrive when the task is not active, they are rejected.

- If the queue contains an interrupt message, it is discarded.
- If the queue contains a user disconnect, user abort, or network abort message, this data is ignored.

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 definition for your language in Section 3.7.1.

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.*

3.7.8 CONNT – Request Logical Link Connection

Use:

Call CONNT from the source task to request a logical link with the target task. Before you call CONNT, you must build a connect block using the BACC and BFMT0 or BFMT1 calls. The BACC call builds the access control information area (unless this data has already been supplied with an alias node name) and the BFMT0 or BFMT1 call builds the destination descriptor format fields. Use BFMT0 when connecting to a task that requires only an object type; use BFMT1 when the task requires only a task name. Refer to Sections 3.7.4 (BACC), 3.7.5 (BFMT0), and 3.7.6 (BFMT1) for full details on each call.

When a remote system receives a connect request, it checks to see if the remote task is currently installed and inactive. If it is, it automatically loads and activates the remote task. The connect block is passed to the remote task (see GNDNT, Section 3.7.11). The task evaluates the connect request and then decides to accept (ACCNT, Section 3.7.3) or reject (REJNT, Section 3.7.14) the request. You can send 1 to 16. bytes/characters of user data to the remote task 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 succeeding 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* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments. If specified, this optional 2-element integer array/data item will contain the following values when the call completes:

First Status Word

status%(0) (BASIC) or

status(1) (FORTRAN, COBOL):

Error/completion code (see list below).

Second Status Word

status%(1) (BASIC) or

status(2) (FORTRAN, COBOL):

Contents depends on error/completion code in first status word (see list below).

Listed below are possible error/completion codes you can receive in the first status word, plus 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 access control information area and destination descriptor. See definition for your language in Section 3.7.1.

outsize,outmessage

defines optional user data you want to send. See definition in Section 3.7.1 and note the exception.

insize,inmessage

defines 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 you want to receive. It must be an integer variable or constant.

* *inmessage* specifies the array/string that will store 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 has been accepted, but some returned optional data was lost (the data sent to the target task when you called CONNT).

-1 System resources needed for the logical link are not available.

-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 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.*

3.7.9 DSCNT – Disconnect a Logical Link

Use:

Call DSCNT from either task to disconnect the logical link and free the logical unit number. When you issue a DSCNT, all pending transmits are completed before the link is disconnected. While these transmits are completing, the task continues to receive messages. When the last transmit has completed, all pending receives are aborted and you receive an abort status in the I/O status block for each one. When you call DSCNT, 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 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 you want 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 definition for your language in Section 3.7.1.

outsize,outmessage

defines optional user data you want to send. See definition in Section 3.7.1.

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.*

3.7.10 GLNNT – Get Local Node Information

Use:

Call GLNNT from either task to have the following local node information placed in the specified buffer:

- Local node name

  You may want to supply the local node name within a program that is to connect to the local node. For example, if you have a program that runs on several nodes and sends data to the local node, your program must supply the local node name to the connect block before it can establish a connection with that node. You also may want to issue a GLNNT within any program that is to display the local node name.

- Default NSP segment size (that is, the size that NSP uses to segment data transmitted on a logical link)

  When you know the default NSP segment size, you can use transmit buffers (large data buffers) most efficiently by adjusting the length of the message blocks to be transmitted.

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 definition for your language in Section 3.7.1.

buflen

specifies an array/string containing the received data length. If you specify 6 bytes/characters, only the local node name will be returned. If you specify 8. bytes/characters, both the node name and the default NSP segment size will be 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)	Content/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.*

3.7.11 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*). If the call completes successfully, the variable specified by the *type* argument will contain a code to indicate which of the following unsolicited message types has been stored:

- 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 will complete with an error (–14).

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 an array/data item containing completion status information on return from GNDNT. In COBOL and BASIC, you cannot omit *status*, but you can specify 0 for *status* if you do not want status information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments. If specified, this optional 2-element integer array/data item will contain the following values when the call completes:

First Status Word

status%(0) (BASIC) or

status(1) (FORTRAN, COBOL):

Error/completion code (see list below).

Second Status Word

status%(1) (BASIC) or

status(2) (FORTRAN, COBOL):

Contents depends on error/completion code in first status word, as described in Status Table A.

Status Table A

If GNDNT completes with an error:

Contents of First Status Word	Contents of Second Status Word
-40	Directive error code
-*n* (other than -40)	0

If GNDNT completes successfully and the *ltonly* flag is -1 (.TRUE.):

Contents of First Status Word	Contents of Second Status Word
+*n*	Low-order byte contains the number of bytes/characters in the first network data item stored in the queue.

If GNDNT completes successfully and the *ltonly* flag is 0 (.FALSE.):

Contents of First Status Word	Contents of Second Status Word
+*n*	Content depends on data message type. Each message type is listed in Status Table B, followed by the contents of the second status word on return from GNDNT.

Status Table B

Contents of Second Status Word

Type Code	Message Type	Low-order Byte	High-order Byte
1	Connect request	Number of bytes/characters in the connect block (the connect block data determines whether request will be accepted (ACCNT) or rejected (REJNT))	Access verification —* and privilege code: 1 = Requesting user is nonprivileged. 2 = Requesting user is privileged. 0 = Verification was not done.—** -1 = Verification failed.—***
2	Interrupt message	Number of bytes/characters in the message. If 0, no message was received.	LUN over which the notice was received.
3	User disconnect		
4	User abort		
5	Network abort	Reason for network abort (see codes in Appendix A).	LUN over which the notice was received.

* If access verification is enabled for the target node, it evaluates access control data in the connect request before it is allowed to pass to the target task's network data queue. For more information on access control, refer to the *DECnet-RSX Network Management Concepts and Procedures* manual.

** Either 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 or verification state for the object is off.

*** 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 containing the data message type code on return from GNDNT. The code indicates the type of data message in the target task's mail buffer on return from GNDNT. Type codes and corresponding message types are listed in Status Table B, above.

mailsz

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

* *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; it can be omitted if you specify *ltonly* as .TRUE. In COBOL and BASIC, it can be set to 0 if you specify *ltonly* as -1.

ltonly

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

If you specify the *typmsk* argument (see below), 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, *ltonly* can be omitted only if all trailing arguments are omitted. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets [] in Section 3.6.

immed

specifies GNDNT action based on data in network data queue.

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

Note that *immed* cannot be omitted in COBOL or BASIC unless all trailing arguments are also omitted. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets [] in Section 3.6.

typmsk

specifies data type to be selected from network data queue. Normally, GNDNT returns items from the network data queue on a first-in, first-out basis. You can specify an integer variable or constant for *typmsk* to select the first item on the queue that matches the message type and/or LUN that you choose.

Table 3–2: Connect Block Received in the Mail Buffer after GNDNT

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)
1.	Destination object type (1–255. for BFMT0, or 0 for BFMT1)
	Descriptor Field for Format 0
18.	Not used
	Descriptor Fields for Format 1
2.	Destination program name length (equal to or less than 16. bytes/characters)
16.	Destination program name

(continued on next page)

Table 3–2 (cont.): Connect Block Received in the Mail Buffer after GNDNT

Length (in decimal bytes/characters)	Contents
	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 (must be either format 0 or format 1)
1.	Source object type (object type of program requesting connection: 1–255. for format 0, or 0 for format 1)
	Descriptor Field for Format 0
18.	Not used
	Descriptor Fields for Format 1
2.	Source program name length (equal to or less than 16. bytes/characters)
16.	Source program name
	ACCESS CONTROL (46.-byte/character total)
	If no verification performed
2.	Source program user ID length (equal to or less than 16. bytes/characters)
16.	Source program user ID
2.	Source program password length (equal to or less than 8. bytes/characters)
8.	Source program password
2.	Account number length (equal to or less than 16. bytes/characters)

(continued on next page)

Table 3–2 (cont.): Connect Block Received in the Mail Buffer after GNDNT

Length (in decimal bytes/characters)	Contents
	ACCESS CONTROL
16.	Account number
	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
	OPTIONAL DATA (18.-byte/character total)
2.	Length of optional user data (equal to or less than 16. bytes/characters; 0 if no optional data)
16.	Optional user data sent by source program (0 to 16. bytes/characters)

Message Type (byte 0)	Logical Unit Number (byte 1)
1 Connect request	0 (Selects the first LUN to request a connect.)
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, if you want to select from the network data queue the first interrupt message (type 2) on LUN 3, you would use a variable for the *typmsk* argument, declare it as an integer, and assign a value to it, as shown here: (3*256.)+2.

If you specify 0 in byte 1, the first message of the type specified in byte 0 will be returned, regardless of the LUN.

NOTE

If you specify *typmsk*, you must also include *mailsz* and *mailbuf*, and you must specify *ltonly* as 0 in COBOL and BASIC, and as .FALSE. in FORTRAN. If you use *typmsk*, you can omit *ltonly* in FORTRAN, but not in COBOL or BASIC.

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 will fit 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.*

3.7.12 OPNNT – Access the Network

Use:

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

Formats:

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

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

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

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 1.2.2). However, when *lun* is omitted in COBOL or BASIC, all trailing arguments must also be omitted. For information on omitting arguments in COBOL and BASIC, refer to the discussion of square brackets [ ] in Section 3.6.

* *status*

specifies completion status information on return from OPNNT. See definition for your language in Section 3.7.1.

* *mstat*

specifies a 3-element integer array (or elementary numeric data item for COBOL) to contain current status information of the task's network data queue. When specified(+), the *mstat* array/data item is updated whenever data arrives or is retrieved by a GNDNT. This array/data item must not be used 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 network data queue

mstat(2) Data type of 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 will accept. 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 will reject all incoming connect requests. You can still establish outgoing links by using CONNT.

lrp

specifies the link recovery period – that is, the number of seconds that can elapse from the time of a physical link failure until the associated logical link is considered 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 will be 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.

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.*

3.7.13 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 data is to be received. 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 definition for your language in Section 3.7.1 and 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 will fit 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.*

3.7.14 REJNT – Reject Logical Link Connect Request

Use:

Call REJNT from the target task to reject a logical link connect request. When you call 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 definition for your language in Section 3.7.1.

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. This array/string is the same one referred to in GNDNT (see Section 3.7.11).

outsize,outmessage

defines optional user data you want to send. See definition in Section 3.7.1.

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.*

3.7.15 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 data is to be sent. 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 SNDNT. See definition for your language in Section 3.7.1 and note this addition: If a 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 you want 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 you want 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.*

3.7.16 WAITNT – Suspend the Calling Task

Use:

Call WAITNT from any task to suspend that task's operation until a call specified by one of the associated status blocks completes.

Formats:

FORTRAN: CALL WAITNT ([*index*],*status*1,...,*statusn*)

COBOL: CALL "WAITNT" USING [*index*],*status*1,...,*statusn*.

BASIC: CALL WAITNT BY REF ([*index*%],*status*1%(),...,*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* if you do not want index information returned. See the discussion of square brackets in Section 3.6 for more information on omitting optional arguments.

*status*1,...,*statusn*

specifies one or more status blocks. WAITNT completes when any one of the calls associated with a status block in this list completes.

3.7.17 XMINT – Send Interrupt Message

Use:

Call XMINT from either task to send an interrupt message over an established logical link. The message you send is placed on the target task's network data queue and must be retrieved with a GNDNT 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 the interrupt message is to be sent. 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 XMINT. See definition for your language in Section 3.7.1.

intsize

specifies the length in bytes/characters of the interrupt message you want to send. It must be an integer variable or constant.

intmsg

specifies a 1- to 16.-element array/string containing the interrupt message you want 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.*

3.7.18 FORTRAN Intertask Communication Programming Example (Transmit)

The following two programs (Transmit and Receive) are examples of FORTRAN intertask communication. They are cooperating tasks. FTNTRN is a transmit task. FTNREC is a receiver task. FTNTRN accesses the network, connects to FTNREC, transmits inquiries to FTNREC, and receives responses from FTNREC.

NOTE

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

```
C
C    TO TASK BUILD USE THE FOLLOWING COMMAND STRING:
C
C   FTNTRN,FTNTRN=FTNTRN,[1,1]NETFOR/LB,F4POTS/LB,RMSLIB/LB
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                    (if RMS included)
C   //
C
        INTEGER*2 MLTYP,RECSIZ,SNDSIZ,OPNLUN,CONLUN,MESNUM,XMITS,NDLEN,TSKLEN
        INTEGER*2 IOST(2),MSTAT(3)
        BYTE ERRMES(2),TSKNAM(6),CONBLK(72),NDNAM(6),DEFNOD(6),DEFTSK(6)
        BYTE SNDBUF(50),RECBUF(10)
        LOGICAL*1 STAT,IMMED
        DATA DEFNOD/'M','A','S','T','E','R'/
        DATA DEFTSK/'R','E','C','V','E','R'/

C
C  INITIALIZE CONSTANTS
C
        IMMED=.TRUE.                    !* SET IMMED TO TRUE FOR GNDNTW
        OPNLUN=1                        !* NETWORK OPNNT LUN
        CONLUN=2                        !* COUNT LUN FOR THE
                                        !* LOGICAL LINK
        XMITS=20                        !* THE NUMBER OF INQUIRIES
                                        !* TO BE SENT TO THE REMOTE NODE
        SNDSIZ=50                       !* THE SIZE OF THE THE MESSAGES TO
                                        !* BE SENT TO THE REMOTE NODE
        RECSIZ=10                       !* THE SIZE OF THE MESSAGES TO
                                        !* BE RECEIVED

C
C  GET THE NODE AND TASK NAMES
C
4       TYPE 300                        !* ASK FOR NODE-NAME
        READ(5,310) (NDNAM(NDLEN),NDLEN=1,6) !* GET THE NAME
        DO 5 NDLEN=6,1,-1               !* LOOP TO FIND LENGTH OF NAME
        IF (NDNAM(NDLEN).NE.' ') GOTO 6 !* IF NOT A SPACE, GET TASK-NAME
5       CONTINUE
        DO 50 I=1,6
50      NDNAM(I)=DEFNOD(I)              !* DEFAULT NODE NAME 'MASTER'
        NDLEN=6                         !* LENGTH OF DEFAULT NAME
```

(continued on next page)

```
6       TYPE 320                        !* ASK FOR THE TASK-NAME
        READ(5,310) (TSKNAM(TSKLEN),TSKLEN=1,6) !* GET IT
        DO 7 TSKLEN=6,1,-1              !* TSKLEN IS LENGH OF TASK-NAME
        IF (TSKNAM(TSKLEN).NE.' ') GOTO 8 !* IF NOT SPACE, ACCESS NETWORK
7       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
C
8       CALL    OPNNTW(OPNLUN,IOST,MSTAT)
        IF      (IOST(1).NE.1)GOTO 100  !* IF FAILURE JUST EXIT
C
C  BUILD A FORMAT 2 CONNECT BLOCK


C
        CALL    BFMT1(STAT,CONBLK,NDLEN,NDNAM,,TSKLEN,TSKNAM)
        IF      (STAT)GOTO 10           !* IF SUCCESS GO ON
        TYPE    200                     !* ELSE TYPE OUT A FAILURE
                                        !* NOTIFICATION
        GOTO    90                      !* AND EXIT
C
C  CONNECT TO THE TASK ON THE REMOTE NODE
C
10      CALL    CONNTW(CONLUN,IOST,CONBLK)
        IF      (IOST(1).EQ.1)GOTO 15   !* IF SUCCESS TELL HIM
        TYPE    240,IOST                !* ELSE PRINT STATUS BLOCK
        GOTO    90                      !* DEACCESS THE NETWORK
                                        !* AND EXIT
15      TYPE    220                     !* PRINT CONNECT CONFIRMATION
                                        !* NETWORK AND EXIT
C
C  SEND AND RECEIVE MESSAGES TO AND FROM THE REMOTE NODE
C
        DO      40 MESNUM=1,XMITS
C
C  FIRST GET ANY ERROR MESSAGES SENT FROM THE OTHER SIDE VIA
C  INTERUPT MESSAGES
        IF      (MSTAT(1).EQ.0)GOTO 20  !* 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 20   !* IF WE COULDN'T GET THE MESSAGE
                                        !* JUST IGNORE IT
        TYPE    210,ERRMES(1)           !* PRINT OUT THE MESSAGE
C
C  SEND THE INQUIRY
C
20      CALL    SNDNTW(CONLUN,IOST,SNDSIZ,SNDBUF)
        IF      (IOST(1).EQ.1)GOTO 30   !* IF SUCCESS CONTINUE
        TYPE    210,MESNUM              !* OTHERWISE TYPE OUT AN
                                        !* ERROR MESSAGE
        GOTO    40                      !* AND START A NEW MESSAGE
C
C  RECEIVE THE RESPONSE FROM THE REMOTE NODE
C
30      CALL    RECNTW(CONLUN,IOST,RECSIZ,RECBUF)
        IF      (IOST(1).EQ.1)GOTO 40   !*IF SUCCESS CONTINUE
        TYPE    210,MESNUM              !* OTHERWISE TYPE OUT AN
                                        !* ERROR MESSAGE
```

(continued on next page)

```
40      CONTINUE
C
C  DISCONNECT THE LINK
C
        TYPE    230                     !* PRINT OUT DISCONNECT MESSAGE
        CALL    DSCNTW(CONLUN,IOST)
C
C  COME HERE TO DEACCESS THE NETWORK AND EXIT
C
90      CALL    CLSNTW
100     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= ',I3)
300     FORMAT  (' PLEASE ENTER NODE-NAME <MASTER>: ',$)
310     FORMAT  (6A1)
320     FORMAT  (' PLEASE ENTER TASK-NAME <RECVER>: ',$)
        END
```

3.7.19 FORTRAN Intertask Communication Programming Example (Receive)

When FTNTRN completes sending inquiries, it disconnects the link, deaccesses the network, and exits. Any error found by FTNREC is transmitted back to FTNTRN as an interrupt message. FTNTRN then displays the interrupt message on the terminal.

```
C
C    TO TASK BUILD USE THE FOLLOWING COMMAND STRING:
C
C   FTNREC,FTNREC=FTNREC,[1,1]NETFOR/LB,F4POTS/LB,RMSLIB/LB
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                  (if RMS included)
C   //
C
        INTEGER*2 OPNLUN,MLTYP,INDEX,ACCLUN,NUMBER,NUMMES
        INTEGER*2 RECSIZ,SNDSIZ,INTSIZ
        INTEGER*2 MSTAT(3),IOST(2),IOST1(2),IOST2(2)
        BYTE RECBUF(50),SNDDAT(10),MLBX(98),INTMES(2)
C
C  INITIALIZE CONSTANTS
C
        OPNLUN=1                 !* NETWORK OPNNT LUN
        ACCLUN=2                 !* ACCNT LUN FOR THE 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
        SNDSIZ=10                !* NUMBER OF BYTES TO SEND BACK
C
C  ACCESS NETWORK
C
        CALL    OPNNTW(OPNLUN,IOST,MSTAT)
        IF      (IOST(1).NE.1)GOTO 100          !* IF FAILURE JUST EXIT
        IF      (MSTAT(1).EQ.0)GOTO 40          !* IF NOTHING ON MAILBOX
                                                !* JUST CLOSE AND EXIT
10      CALL    GNDNT(IOST1,MLTYP,98,MLBX)      !* ISSUE A 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  NETWORK DATA HAS BEEN RECEIVED
C
        IF      (IOST1(1).NE.1)GOTO 40          !* IF GNDNT FAILED JUST
                                                !* CLOSE AND EXIT
        IF      (MLTYP.GE.3)GOTO 40             !* IF MULTYP>=3 THE LINK HAS
                                                !* BEEN BROKEN
        IF      (MLTYP.EQ.2)GOTO 10             !* IF MLTYP=2 WE'VE RECEIVED
                                                !* AN INTERRUPT MESSAGE, JUST
                                                !* ISSUE A NEW GNDNT
C
C  WE'VE RECEIVED A CONNECT REQUEST - ISSUE AN ACCEPT
C
        CALL    ACCNTW(ACCLUN,IOST,MLBX)
        IF      (IOST(1).NE.1)GOTO 10           !* IF FAILURE ISSUE A NEW
                                                !* GNDNT
```

(continued on next page)

```
C
C  ISSUE A RECEIVE TO PICK UP DATA
C
30      CALL    RECNT(ACCLUN,IOST2,RECSIZ,RECBUF)
        GOTO     10                                  !* ISSUE A NEW GNDNT
                                                     !* AND WAIT FOR A COMPLETION
C
C  WE COME HERE UPON RECEIVING A DISCONNECT OR ABORT
C
40      CALL    CLSNTW                               !* DEACCESS THE NETWORK
        GOTO    100                                  !* AND EXIT

C
C  WE COME HERE IF WE RECEIVE AN INQUIRY
C
50      NUMMES=NUMMES+1                              !* INCREMENT THE MESSAGE
                                                     !* COUNT
        IF      (IOST2(1).EQ.1)GOTO 60               !* IF IOST2(1)-1 ALL'S O.K.
C
C  IF THERE WAS AN ERROR SEND BACK AN INTERRUPT MESSAGE WITH
C  MESSAGE NUMBER
C
        INTMES(1)=NUMMES                             !* SEND THE MESSAGE NUMBER
        CALL    XMINT(ACCLUN,IOST,INTSIZ,INTMES)
        GOTO    70                                   !* GO ISSUE A NEW RECEIVE
C
C  HERE THE USER CAN LOOK AT THE DATA RECEIVED IN RECBUF AND RESPOND
C  BY PLACING THE REQUESTED INFORMATION INTO SNDDAT
C
C
C  SEND BACK THE DATA AND ISSUE A NEW RECNT
C
60      CALL    SNDNTW(ACCLUN,IOST,SNDSIZ,SNDDAT)
70      CALL    RECNT(ACCLUN,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
```

3.7.20 COBOL Intertask Communication Programming Example (Transmit)

The following two programs (Transmit and Receive) are examples of COBOL intertask communication. They are cooperating tasks. COBTRN is a transmit task. COBREC is a receiver task. COBTRN accesses the network, connects to COBREC, transmits inquiries, and receives responses from COBREC.

NOTE

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

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBTRN.

*****************************************************************************
*                                                                           *
*       THIS IS THE TRANSMIT PROGRAM OF THE DECNET  COBOL                   *
*       INTERFACE COMMUNICATION EXAMPLE PROGRAMS.                           *
*                                                                           *
*    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 included)                           *
*   //                                                                      *
*                                                                           *
*****************************************************************************

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.
  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)
```

(continued on next page)

```
-                                       " = ".
        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  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 S9      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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       ACCESS THE NETWORK.  IF THE ACCESS IS UNSUCCESSFUL,     *
*       PRINT AN ERROR MESSAGE AND EXIT.                        *
*                                                               *
*****************************************************************

        CALL "OPNNTW" USING
                OPNLUN
                IOST
                MSTAT
                TEN.
        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
                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 UNSUCESSFULLY, 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".
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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.                            *
*                                                               *
*****************************************************************
```

(continued on next page)

```
        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.

*****************************************************************
*                                                               *
*       DEACESS 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.
```

3.7.21 COBOL Intertask Communication Programming Example (Receive)

When COBTRN completes sending inquiries, it disconnects the link, deaccesses the network, and exits. Any error found by COBREC is transmitted to COBTRN as an interrupt message. COBTRN then displays the interrupt message on the terminal.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBREC.

*****************************************************************************
*                                                                           *
*       THIS IS THE RECEIVE PROGRAM OF THE DECNET COBOL                     *
*       INTERFACE COMMUNICATION EXAMPLE PROGRAMS.                           *
*                                                                           *
*    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 included)                          *
*   //                                                                      *
*                                                                           *
*                                                                           *
*****************************************************************************

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.
   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.
```

(continued on next page)

```
01  STORE-STUFF.
    03  OPNLUN          PIC 99       COMP VALUE 2.
    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.
        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.                                            *
*                                                               *
*****************************************************************
```

(continued on next page)

```
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
                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.                            *
*                                                               *
*****************************************************************
```

(continued on next page)

```
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.
```

3.7.22 BASIC–PLUS–2 Intertask Communication Programming Example (Transmit)

The following two programs (Transmit and Receive) are examples of BASIC–PLUS–2 intertask communication. They are cooperating tasks. BASTRN is a transmit task. BASREC is a receiver task. BASTRN accesses the network, connects to BASREC, transmits inquiries to BASREC, and receives responses from BASREC.

NOTE

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

```
10      !!!    To task build you must 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


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
```

(continued on next page)

```
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 BE SENT TO THE REMOTE NODE  &
 \      SNDSIZ%=50%                     !THE SIZE OF THE MESSAGES TO    &
                                        !BE SENT TO THE REMOTE NODE     &
 \      RECSIZ%=10%                     !THE SIZE OF THE MESSAGES TO    &
                                        !BE RECEIVED                    &
 \      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 HIM            &
        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 COMFIRMATION     &
                                        !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 INTERUPT 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
```

(continued on next page)

```
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
```

3.7.23 BASIC–PLUS–2 Intertask Communication Programming Example (Receive)

When BASTRN completes sending inquiries, it disconnects the link, deaccesses the network, and exits. Any error found by BASREC is transmitted to BASTRN as an interrupt message. BASTRN then displays the interrupt message on the terminal.

```
10      !!!    To task build you must 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 THE 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 A 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
```

(continued on next page)

```
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 O.K.

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$)

130     CALL    RECNT BY REF(ACCLUN%,IOST2%(),RECSIZ%,RECBUF$)          &
 \      GOTO    50                      !WAIT FOR A COMPLETION

140     !!! EXIT PROGRAM !!!                                            &
        PRINT   "END OF PROGRAM EXECUTION"                              &
 \      END
```

3.8 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 Call Summary

Call	Function
ACONFW	To set record and file access options
ATTNFW	To set extended attributes
CLSNFW	To close a file
DELNFW	To delete a file
EXENFW	To execute a file
GETNFW	To read a single record
OPANFW	To open and append a sequential file
OPMNFW	To open and modify a sequential file
OPRNFW	To open and read a sequential file
OPUNFW	To open and update a sequential file
OPWNFW	To create, open, and write a sequential file
PRGNFW	To discard an open file
PUTNFW	To write a record to a file
RENNFW	To rename a file
SPLNFW	To open, write, and print a file
SUBNFW	To 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] allows your task to access the network. Issue CLSNT[W] last to close your task's access to the network.

3.8.1 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 be one not in use.

You must 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.

3.8.2 Performing File Operations

The following subroutines perform file operations:

EXENFW Executes a remote file

DELNFW Deletes a remote file

RENNFW Renames a remote file

3.8.3 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

3.8.4 Closing Files and Completing Calls

When you complete a file access operation, use CLSNFW to close the file. If you want 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 deaccess the network, or if a network abort occurs while the file is open, the network will close 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.

3.8.5 Setting Task Build Parameters

DECnet-RSX uses network file access routines (NFARs) as the interface at the local node to access remote files for user applications. At task build time you can override defaults to tailor these NFARs for a particular application. Task build parameters you can set and their defaults are:

- Event flags .TREF(default 17) and .RCEF(default 18)
- Buffering level (default 2)
- Maximum record size (default 256. bytes)
- Buffer space allocation (no default)

3.8.5.1 Setting Event Flags — The network file access routines (NFARs) require the exclusive use of two event flags. By default, the event flags used 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.

3.8.5.2 Setting Buffering Level — The NFARs can be configured for multibuffering to improve throughput. However, this requires more internal buffering space. The default buffering level is 2. To override this default, issue the following command in the task builder command file:

GBLDEF=$NFRSZ:*buffering-level*

The *buffering-level* variable specifies an integer from 1 to 4. If the remote system is RSX or IAS, you can ask the system manager for the buffering level used for that system and use the same one.

3.8.5.3 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. If the remote system is RSX or IAS, use the same record size used by the remote system.

If you intend to transfer an ASCII file, add an extra 2 bytes when calculating the maximum record size. The extra 2 bytes are required for the carriage return and line feed characters that are appended to each ASCII record. If you intend to transfer sequenced variable length records, add an extra 2 bytes for the sequence number included with each record. If you intend to transfer an ASCII file with sequenced variable length records, add an extra 4 bytes.

3.8.5.4 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. Note that the module NFAFSR from the NETFOR object library must be included in your task. 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 amount of buffer space required to perform remote file access:

(($NFRSZ+14.)*($NFNMB+1))+64.)*(*max-rem-files*)+(512.*<*max-loc-files*>)

where

max-rem-files is the maximum number of remote files that can be opened simultaneously.

max-loc-files is the maximum number of local files that can be opened simultaneously.

Note that the amount of space (512.) needed for local files will vary according to the language you are using.

At task build time, extend the file storage region by the amount of space calculated by this formula by including the following command in the task builder command file (the value given must be in octal bytes):

EXTSCT=$$FSR1:*value*

3.8.5.5 Using the Task Build Procedure — It is necessary for a task to link to NETFOR.OLB in order to use the DECnet-RSX remote file access capabilities. Edit the ODL file created by the compiler. The following is an example of 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 edits required for the task builder to include remote file access capabilities in the task. Boldface items are necessary 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
```

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:.FCTRLB:[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
```

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
```

3.8.6 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, 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$)
```

3.8.7 Common Argument Definitions for Remote File Access Calls

Arguments commonly used for remote file access calls are defined on the following pages to avoid needless repetition throughout the call descriptions. Argument definitions here are divided into four categories: a general category containing argument definitions common to all languages, and three individual language categories for arguments with language-specific definitions.

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 node to which the call is directed. 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 43.-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 will contain completion status information on return from the call. This 2-element single-precision integer array will contain the following values when the call completes:

  status(1) returns an error completion code. Refer to Table C–1 in Appendix C.

  status(2) depends on the content 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 will contain completion status information on return from the call. This elementary numeric data item will contain the following values when the call completes:

  status(1) returns an error completion code. Refer to Table C–1 in Appendix C.

  status(2) depends on the content of the first status word. Refer to Appendix C.

BASIC-PLUS-2

- *status*%()

  specifies an array that will contain completion status information on return from the call. This 2-element integer array will contain the following values when the call completes:

  status%(0) returns an error completion code. Refer to Table C–1 in Appendix C.

  status%(1) depends on the content of the first status word. Refer to Appendix C.

3.8.8 ACONFW – Set Access Options

Use:

Call ACONFW before a specific file operation to specify record and file access options you want applied to that file operation. These options remain in effect until the file is closed.

Formats:

FORTRAN: CALL ACONFW (*lun*,[*fac*],[*shr*],[*fop*],[*accopt*)

COBOL: CALL "ACONFW" USING *lun*,[*fac*],[*shr*],[*fop*].

BASIC: CALL ACONFW BY REF (*lun*%,[*fac*%],[*shr*%],[*fop*%()])

Arguments:

lun

specifies the logical unit number of the logical link assigned to the file operation you want to set options for.

fac

specifies the file access operations to be allowed while accessing the file. Use this argument only for open and create operations. The *fac* value overrides the specific OP*x*NFW call used. Valid values are:

NF$PUT	Put access
NF$GET	Get access (default)
NF$DEL	Delete record access
NF$UPD	Update record access
NF$TRN	Truncate file access
NF$BIO	Block I/O
NF$REA	Block I/O read
NF$WRT	Block I/O write

shr

specifies the file sharing to be allowed by the remote system. Use this argument only for open and create operations. The actual functioning is dependent on remote system capabilities. Valid values are:

NF$PUT	Put access
NF$GET	Get access (default)
NF$DEL	Delete record access
NF$UPD	Update record access
NF$NIL	No access to others

* *fop*

specifies a 3-word array for file-processing options to be used for open, create, and close operations. To specify a *fop* value for close operations, the ACONFW call must be made after the file has been opened, because the open call will overwrite the current value. Valid values are:

First word:

NF$CTG	Create contiguous file
NF$SUP	Supercede existing file
NF$TMP	Create temporary file
NF$MKD	Create temporary file and mark for delete

Second word:

NF$MXV	Maximize version number on create
NF$SPL	Spool on close
NF$EXC	Execute on close
NF$DLC	Delete on close
NF$TEF	Truncate on close

Completion of open/create operations returns:

First word:

NF$FLK	File is locked
NF$CTG	File is contiguous
NF$DIR	File is a directory (system dependent)

3.8.9 ATTNFW – Set Extended Attributes

Use:

Use ATTNFW to specify extended attributes to be used by create, open, and close operations. These attributes will be returned to the caller after the specified operation completes.

Call ATTNFW immediately before file create operations (OPWNFW, SPLNFW, and SUBNFW) to specify additional attributes when creating the file.

Call ATTNFW immediately before file open operations (OPRNFW, OPANFW, and RENNFW) to specify additional attributes to be returned when opening the file.

Call ATTNFW immediately before file close operations to specify a change-attributes-on-close sequence.

Formats:

FORTRAN: CALL ATTNFW (*lun*,[*namesize*],[*name*],[*atb*],
[*protblk*],[*owner*],[*datemenu*],[*dateblk*])

COBOL: CALL "ATTNFW" USING *lun*,[*namesize*],[*name*],[*atb*],[*protblk*],
[*owner*],[*datemenu*],[*dateblk*].

BASIC: CALL ATTNFW BY REF (*lun*%,[*namesize*%],[*name*$],[*atb*%()],
[*protblk*%()],[*owner*$],[*datemenu*%],
[*dateblk*%()])

Arguments:

lun

specifies the logical unit number of the logical link for the specified file operation.

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 will be an ASCIZ string.

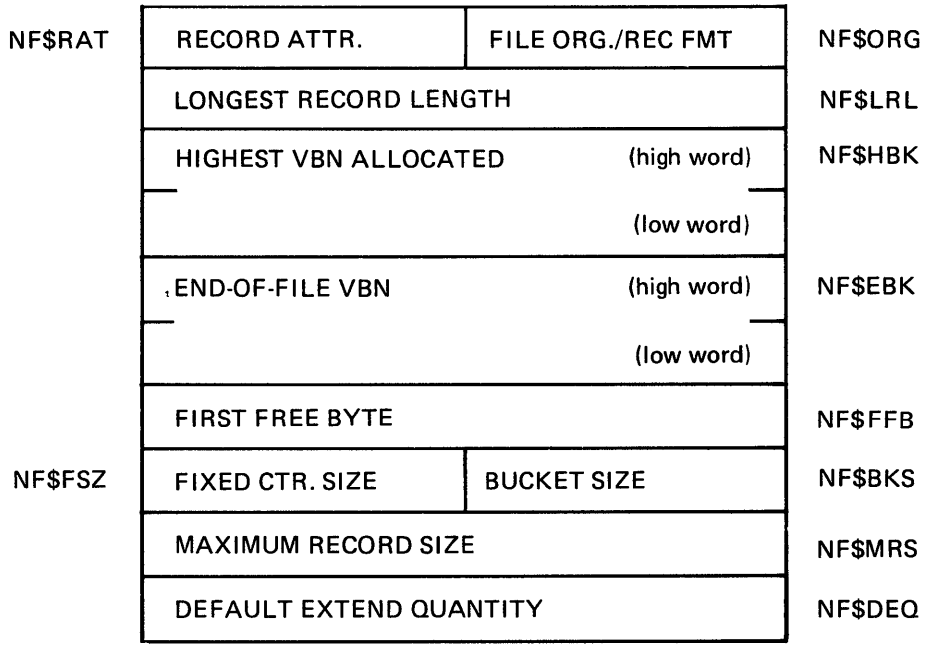
* *atb*

specifies a Files-11 user file attributes block. When specifying create operations, the user program is responsible for setting valid values because the NFARs will not check these values.

NOTE

If *atb* is specified, the *ichar*(2), *ichar*(3), and *len* arguments will be 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:



For further information on the values of these fields, see the *IAS/RSX I/O Operations Reference Manual*, Appendixes A and F.

* *protblk*

specifies an array containing file protection information to be used for input for create and close operations, and returns information from create and open operations. The following format describes a 5-word array:

FILE OWNER STRING SIZE
SYSTEM PROTECTION MASK
OWNER PROTECTION MASK
GROUP PROTECTION MASK
WORLD PROTECTION MASK

If the file owner size is 0, the owner string will not be returned.

The format of the protection masks is described 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 8 = Deny change protection access
Bit 9 = Deny extend access

If a word is set to a –1, that protection will not be sent.

* *owner*

specifies an ASCIZ string/array identifying the file owner to be used for input for create operations and for output from create and open operations. The maximum size of this string/array must be specified in the first word of the *protblk* array.

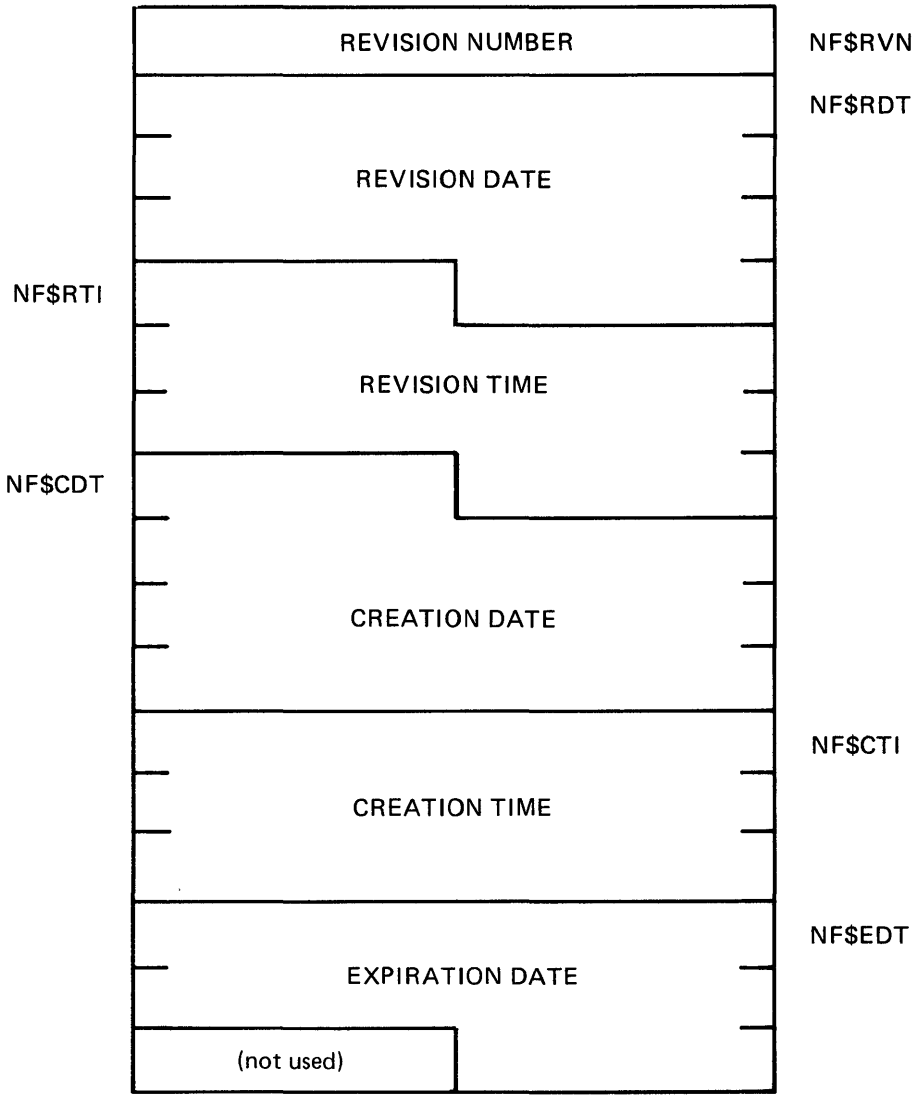
* *datemenu*

specifies which values in the date block are to be used for input for create and close operations or those values returned on output from create and open operations. Valid values are:

Bit 0 = Revision number
Bit 1 = Revision date and time
Bit 2 = Creation date and time
Bit 3 = Expiration date

* *dateblk*

specifies an 18-word array containing the date, time, and revision information associated with the file. Files-11 timestamps are stored in ASCII in the format DDMMMYYHHMMSS, and have the following restrictions: leading zeros are shown, and MMM is the 3-letter month abbreviation in uppercase letters. The following format shows dates in a block:



For further information on the format of dates in a block, see the *IAS/RSX I/O Operations Reference Manual* Appendix F.

3.8.10 CLSNFW – Close a File

Use:

Call CLSNFW to close a remote file. CLSNFW forces completion of all pending file operations, ensures the file directory information is valid, optionally modifies the attributes specified by the *changeattr* argument and a previous ATTNFW call, and frees the logical unit number when the call completes.

NOTE

Some systems do not support the change attribute on close capability.

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 you want to close. See definition in Section 3.8.7. Use the same LUN specified in the previous open call.

* *status*

specifies completion status information on return from CLSNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

changeattr

specifies the attributes to be changed when this file is closed. The attributes must have been specified by a previous ATTNFW call either at open time or just prior to this call. Valid values are:

NF$PRC	Change protection
NF$DTC	Change dates and times

3.8.11 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 you want to delete. See definition in Section 3.8.7.

* *status*

specifies completion status information on return from DELNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file you want to delete. See definition in Section 3.8.7.

ident

is an array/string containing access control information. See definition in Section 3.8.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be deleted. See definition in Section 3.8.7.

3.8.12 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 completion of this call.

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 you want to execute. See definition in Section 3.8.7.

* *status*

specifies completion status information on return from EXENFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file you want to execute. See definition in Section 3.8.7.

ident

is an array/string containing access control information. See definition in Section 3.8.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be executed. See definition in Section 3.8.7.

3.8.13 GETNFW – Read a Single Record

Use:

Call GETNFW to read a record from a file. The record is stored in an array/string that you specify in the *inarray* argument for FORTRAN programmers or the *instring* argument for COBOL and BASIC programmers. Each successive GETNFW call reads the record into the same array/string. The previous record is overlaid and is no longer available in the user record storage area.

If the optional *rac* argument is not specified, the default will be sequential record transfer (NR$RTM). The records will be read sequentially from the first record in the file.

If a *rac* argument is given and specifies random access, the *keyptr* argument must specify the record to be 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 definition in Section 3.8.7. 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 definition for your language in Section 3.8.7. 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 will contain the record to be 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 be 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 record access mode to be used while accessing the file. Use this single-word argument only for open/create operations. Once a file transfer mode is selected, the *rac* value will be ignored until the end-of-file is detected. Valid values are:

NR$SEQ	Sequential by record
NR$KEY	Random by relative record number (RRN)
NR$RFA	Random by record file address (RFA)
NR$RTM	Sequential file transfer by records (default)
NR$VBN	Random blocks by virtual block number (VBN)
NR$BTM	Sequential file transfer by blocks

keyptr

specifies the record. The length is assumed from the *rac* argument value.

RAC	Key
NR$KEY	Two-word binary value of RRN Low-order word first
NR$RFA	Three-word binary RFA Low-order word first
NR$VBN	Two-word binary value of VBN Low-order word first

rop

specifies record processing options. Valid values are:

NR$EOF	Position to EOF
NR$UIF	Update if existing record

3.8.14 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

OPWNFW - Create and Open a File for Writing Records

See Section 3.8.18.

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.

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 definition in Section 3.8.7.

* *status*

specifies completion status information on return from OPANFW, OPMNFW, OPRNFW, or OPUNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file you want to open. See definition in Section 3.8.7.

ident

is an array/string containing access control information. See definition in Section 3.8.7.

ifile

specifies an ASCIZ string containing the file specification for the file to be opened. See definition in Section 3.8.7.

* *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)

3.8.15 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 you want to close. See definition in Section 3.8.7. Use the same LUN specified in the previous open call.

* *status*

specifies completion status information on return from PRGNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

3.8.16 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.

If the optional *rac* argument is not specified, the default will be sequential record transfer (NR$RTM). The records will be written sequentially beginning at the first record position unless the file was opened with an OPANFW call, in which case they are written after the last record.

If a *rac* value is given and specifies random access, the *keyptr* argument must specify the record position where the record will be written.

If a PUTNFW call returns an error, you can close the output file with either a PRGNFW or CLSNFW call, or you can 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 definition in Section 3.8.7. 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 definition for your language in Section 3.8.7. 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/outchars* 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 record access mode to be used while accessing the file. Use this single-word argument only for open/create operations. Once a file transfer mode is selected, the *rac* value will be ignored until after end-of-file is detected. Valid values are:

NR$SEQ	Sequential by record
NR$KEY	Random by relative record number (RRN)
NR$RFA	Random by record file address (RFA)
NR$RTM	Sequential file transfer by records (default)
NR$VBN	Random blocks by virtual block number (VBN)
NR$BTM	Sequential file transfer by blocks

keyptr

specifies the record. The length is assumed from the *rac* argument value.

RAC	Key
NR$KEY	Two-word binary value of RRN Low-order word first
NR$RFA	Three-word binary RFA Low-order word first
NR$VBN	Two-word binary value of VBN Low-order word first

rop

specifies record-processing options. Valid values are:

NR$EOF	Position to EOF
NR$UIF	Update if existing record

3.8.17 RENNFW – Rename a File

Use:

Call RENNFW to rename a remote file.

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 created for renaming a remote file. See definition in Section 3.8.7.

* *status*

specifies completion status information on return from RENNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file you want to rename. See definition in Section 3.8.7.

ident

is an array/string containing access control information. See definition in Section 3.8.7.

ofile

specifies an ASCIZ array/string containing the file to be renamed.

nfile

specifies an ASCIZ array/string containing the new file specification for the newly renamed file.

NOTE

If a name buffer is attached via ATTNFW, the resultant file specification of the new file will be returned in that name buffer.

3.8.18 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 a new remote file, write to it, and print it at the remote node.

Call SUBNFW to create a new remote file, write to it, and submit it to the remote batch/command file processor for execution. The file is deleted after execution.

Successful completion of SUBNFW does not mean that the file ran properly or even that it ran at all. Successful completion of this call implies only that the file was handled properly by the remote node.

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 definition in Section 3.8.7.

* *status*

specifies status completion information on return from SPLNFW, SUBNFW, or OPWNFW. See definition for your language in Section 3.8.7. Refer to Table C–1 in Appendix C for a complete code list.

node

specifies the name of the node for the file you want to open using SPLNFW, SUBNFW or OPWNFW. See definition in Section 3.8.7.

ident

is an array/string containing access control information. See definition in Section 3.8.7.

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 you want 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.

3.8.19 FORTRAN Remote File Access Programming Example (Append)

The program FTNAPP 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 is FRED, the password is PRIV, and the account number is 1.

NOTE

This programming example is also included in your tape or disk kit.

```
C
C       THIS PROGRAM IS A VARIATION OF THE PROGRAM THAT APPEARS IN THE
C       RSX DECNET PROGRAMMER'S REFERENCE MANUAL
C
C     TO TASK BUILD USE THE FOLLOWING COMMAND STRING:
C
C    FTNAPP,FTNAPP=FTNAPP,[1,1]NETFOR/LB,F4POTS/LB,RMSLIB/LB,NETFOR/LB:NFAFSR
C    /
C    UNITS=10
C    EXTSCT=$$FSR1:2700
C    ACTFIL=4
C    EXTTSK=1000                    (if RMS included)
C    //
C
C
        BYTE NODE(7),BUFFER(128),IDENT(30),ICHAR(3)
        INTEGER ISTAT(2),JSTAT(2),KSTAT(2),LSTAT(2),MSTAT(3),NSTAT(2)
        DATA ICHAR/'A','V','F'/
C
C       GET USER ID, PASSWORD AND ACCOUNT
C
        TYPE 666
666     FORMAT ('$ENTER USER I.D.: ')
        ACCEPT 779,ICNT,(IDENT(I),I=1,ICNT)
        IDENT(ICNT+1)=0
        TYPE 667
667     FORMAT ('$ENTER PASSWORD: ')
        ACCEPT 779,ICNT9,(IDENT(I),I=ICNT+2,ICNT+2+ICNT9-1)
        IDENT(ICNT+2+ICNT9)=0
        TYPE 668
668     FORMAT ('$ENTER ACCOUNT NUMBER: ')
        K=ICNT+1+ICNT9+2
        ACCEPT 779,ICNT8,(IDENT(I),I=K,K+ICNT8-1)
779     FORMAT (Q,10A1)
        TYPE *, 'IDENT = ',(IDENT(I),I=1,15)
        TYPE *, (IDENT(I),I=16,30)
C
C       GET REMOTE NODE NAME LOCAL AND REMOTE FILE SPECS
C
95      TYPE 100
100     FORMAT ('$ENTER REMOTE NODE NAME (6 CHAR. MAX.):')
        ACCEPT 110,ICNT3,(NODE(I),I=1,ICNT3)
110     FORMAT (Q,6A1)
        IF (ICNT3-6) 115,115,95
115     TYPE *,'NODE NAME =',(NODE(I),I=1,7)
        TYPE 120
120     FORMAT ('$ENTER FILE SPEC. OF REMOTE FILE FOR APPEND:')
        ACCEPT 130,ICNT1,(BUFFER(I),I=1,ICNT1)
130     FORMAT (Q,64A1)
        BUFFER(ICNT1+1)=0
        TYPE 140
```

(continued on next page)

```
140     FORMAT ('$ENTER FILE SPEC. OF LOCAL FILE TO BE APPENDED:')
        ACCEPT 150,ICNT2,(BUFFER(I),I=64,63+ICNT2)
150     FORMAT (Q,64A1)
        BUFFER(ICNT2+64)=0
C
C       CREATE NETWORK MAILBOX FOR ONLY ONE LINK
C
        CALL OPNNTW (2,LSTAT,MSTAT,1)
        IF (LSTAT(1)-1) 907,160,907
C
C       OPEN LOCAL AND REMOTE FILES
C
160     OPEN (UNIT=4,NAME=BUFFER(64),TYPE='OLD',ERR=901)
        CALL OPANFW (1,ISTAT,NODE,IDENT,BUFFER,ICHAR,LENGTH,IBLOCK)
        IF (ISTAT(1)-1) 908,200,908
C
C       READ RECORDS FROM LOCAL FILE AND WRITE THEM TO REMOTE FILE
C
200     READ (4,210,END=300,ERR=902) ICNT3,(BUFFER(I),I=1,ICNT3)
210     FORMAT (Q,128A1)
        CALL PUTNFW (1,JSTAT,ICNT3,BUFFER)
        IF (JSTAT(1)-1) 903,200,903
C
C       EOF FOUND -- CLOSE BOTH FILES AND NET
C
300     CLOSE (UNIT=4,ERR=904)
        CALL CLSNFW (1,KSTAT)
        IF (KSTAT(1)-1) 905,310,905
310     CALL CLSNTW (NSTAT)
        IF (NSTAT(1)-1) 906,320,906
320     STOP 'APPEND O.K.'
C
C       ERROR HALTS
C
901     TYPE *, (BUFFER(I),I=1,63)
        TYPE *, (BUFFER(I),I=64,128)
        STOP 'CAN NOT OPEN LOCAL FILE'
902     STOP 'READ ERROR FROM LOCAL FILE'
903     TYPE *,'STATUS =',JSTAT(1)
        STOP 'WRITE ERROR FROM REMOTE FILE'
904     STOP 'CAN NOT CLOSE LOCAL FILE'
905     TYPE *,'STATUS =',KSTAT(1)
        STOP 'CAN NOT CLOSE REMOTE FILE'
906     STOP 'CAN NOT CLOSE NETWORK'
907     TYPE *,'STATUS =', LSTAT(1)
        STOP 'MAILBOX CREATION ERROR'
908     TYPE *,'STATUS =' ,ISTAT(1),ISTAT(2)
        STOP 'CAN NOT OPEN REMOTE FILE'
C
        END
```

3.8.20 FORTRAN Remote File Access Programming Example (Read/Write)

The program FTNRRW reads the contents of one remote file into another remote file. When an end-of-file is encountered, the last record is written to the remote file and both files are closed. If a read or write error occurs a message is displayed and the program exits. In the following example, the user ID is JOE, the password is PRIV, and the account number is 404.

NOTE

This programming example is also included in your tape or disk kit.

```
C       read from a remote file and write to a remote file using
C       decnet fortran remote file access support
C
C    To task build use the following command string:
C
C   FTNRRW,FTNRRW=FTNRRW,[1,1]NETFOR/LB,F4POTS/LB,RMSLIB/LB,NETFOR/LB:NFAFSR
C   /
C   UNITS=10
C   EXTSCT=$$FSR1:10000
C   ACTFIL=4
C   EXTTSK=1000                  (if RMS included)
C   //
C
C
C       declare the necessary data structures

        DIMENSION       IARRAY(256),ISTAT(2),IBLK(1),LNTH(1)
        COMMON          ISTAT
        LOGICAL*1 IDINFO(13),ICHARS(3)
        LOGICAL EOF
        DATA EOF/.FALSE./

C       ASCIZ strings for user ID, password and account number.

        DATA IDINFO/'J','O','E',0,'P','R','I','V',0,'4','0','4',0/

C       array containing mode, record type and carriage control information

        DATA ICHARS/'I','V','T'/

c       declare network task

        CALL    OPNNTW(7,ISTAT,,2)
        CALL    CKSTAT

C       open two files, one for input and one for output.
C       both of these files exist on a remote node.

C       open file for input

        CALL    OPRNFW(1,ISTAT,'IASNOD',IDINFO,'
     X  [133,224]NET.TST',ICHARS,LNTH,)
        CALL    CKSTAT
```

(continued on next page)

```
C       open file for output

        CALL    OPWNFW(2,ISTAT,'IASNOD',IDINFO,'
     X  [133,224]NEWNET.TST',ICHARS,LNTH,)
        CALL    CKSTAT

C       once the files are successfully opened, we may transfer records.
C       the file associated with lun 1 is opened for reading, the
C       file on lun 2 is opened for writing.

C       transfer files
        DO 30 I=1,100,1

C       get a record from the input file.

        CALL    GETNFW(1,ISTAT,256,IARRAY)
C       status code 050047 is end of file

        IF (ISTAT(1) .NE. 1 .AND. ISTAT(2) .NE. '050047'O) GO TO 40
        LNTH(1) = ISTAT(2)

C       check for end of file

        IF (ISTAT(1) .NE. 1 .AND. ISTAT(2) .EQ. '050047'O) EOF=.TRUE.
        IF      (EOF .EQ. .TRUE.) GO TO 50
        CALL    PUTNFW(2,ISTAT,LNTH,IARRAY)
        IF (ISTAT(1) .NE. 1 .AND. ISTAT(2) .NE. '050047'O) GO TO 40
30      CONTINUE
40      PRINT 41
41      FORMAT (1H1,'READ OR WRITE ERROR OCCURRED')

C       close files
C       now that files have been transferred we may close
C       the files

50      DO 55 I=1,2,1
        CALL    CLSNFW(I,ISTAT)
55      CONTINUE

        IF      (EOF .NE. .TRUE.) GO TO 60
        PRINT   57
57      FORMAT(1H1,'END OF FILE REACHED, FILES CLOSED')

60      STOP
        END

        SUBROUTINE      CKSTAT
        DIMENSION       ISTAT(2)
        COMMON          ISTAT

        IF      (ISTAT(1) .EQ.1) GO TO 10
5       FORMAT(1H1,'OPEN ERROR')
        PRINT   5
        STOP
10      RETURN
        END
```

3.8.21 COBOL Remote File Access Programming Example (Append)

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.

NOTE

This programming example is also included in your tape or disk kit.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBAPP.

*****************************************************************************
*                                                                           *
*       THIS PROGRAM APPENDS THE CONTENTS OF A LOCAL ASCII                  *
*       FILE TO THE END OF 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 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
```

(continued on next page)

```
-                           "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".
01  NULL1                  PIC 9         COMP VALUE 0.
01  NULLS REDEFINES NULL1.
    03 NUL OCCURS 2 TIMES   PIC X(1).
PROCEDURE DIVISION.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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
*****************************************************************
*                                                               *
*       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.
```

3.8.22 COBOL Remote File Access Programming Example (Read/Write)

The program COBRRW reads the contents of one remote file into another remote file. When an end-of-file is encountered, the last record is written to the remote file and both files are closed.

NOTE

This programming example is also included in your tape or disk kit.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBRRW.

*****************************************************************************
*                                                                           *
*       THIS PROGRAM READS THE CONTENTS OF A REMOTE FILE                    *
*       INTO ANOTHER REMOTE FILE.  EACH RECORD IS READ AND                  *
*       WRITTEN UNTIL END-OF-FILE OCCURS AT WHICH TIME THE                  *
*       LAST RECORD IS WRITTEN TO THE REMOTE FILE AND                       *
*       BOTH FILES ARE CLOSED.                                              *
*                                                                           *
*    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 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
```

(continued on next page)

```
-                               "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 S9         USAGE COMP.
    03  TTRUE                  PIC S9         COMP VALUE -1.
    03  FFALSE                 PIC S9         COMP VALUE 0.
    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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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
                MOVE IOSTAT (2) TO MSG2-STAT2
                DISPLAY MSG2
                GO E100-END.

*****************************************************************
*                                                               *
*       OPEN REMOTE FILE FOR OUTPUT.  IF THERE IS AND 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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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.
```

(continued on next page)

```
*****************************************************************
*                                                               *
*       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.
```

3.8.23 BASIC–PLUS–2 Remote File Access Programming Example (Append)

The program BASAPP 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.

NOTE

This programming example is also included in your tape or disk kit.

```
5       !!!    To task build you must edit the task build command   !!! &
        !!!    file and the ODL file created by the build.          !!! &

        !!!     >Add the lines                                      !!! &
        !!!                                                         !!! &
        !!!             ACTFIL=4                                    !!! &
        !!!             EXTSCT=$$FSR1:2700                          !!! &
        !!!     to the task build command file.                     !!! &
        !!!                                                         !!! &
        !!!     >Append                                             !!! &
        !!!                                                         !!! &
        !!!             -NETLIB-NETLB2                              !!! &
        !!!     to the USER: line of the ODL file.                  !!! &
        !!!                                                         !!! &
        !!!     >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 145              !ERROR HANDLER

10      !!! DEFINE ARRAY CONTSTANTS !!!                                 &
 \      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

15      !!! DEFINE CONSTANTS !!!                                        &
        OPNLUN%=2%                      !NETWORK OPEN LUN               &
 \      APPLUN%=1%                      !FILE LUN                       &
 \      COUNT%=1%                       !MAX # OF LOGICAL LINKS         &
 \      FLAG%=0%                        !END OF FILE FLAG               &
 \      ICHAR$="AVF"                    !MODE, TYPE, CARRIAGE CONTROL

20      INPUT "ENTER REMOTE NODE NAME (MAX. 6 CHARACTERS)";NODNAM$      &
 \      IF      LEN(NODNAM$)>6% THEN PRINT                              &
                "NAME TOO LONG, PLEASE RE-ENTER"                        &
 \              PRINT \ GO TO 20

30      PRINT "NODE NAME = ";NODNAM$ \ NODNAM$=NODNAM$+NULL$            &
                                        !CREATE ASCIZ STRING FOR OPANFW

40      PRINT   "ENTER FILE SPEC. OF REMOTE FILE FOR APPEND:"           &
 \      INPUT   BUF1$ \ BUF1$=BUF1$+NULL$                               &
 \      PRINT   "ENTER FILE SPEC. OF LOCAL FILE TO BE APPENDED:"        &
 \      INPUT   BUF2$
```

(continued on next page)

```
50      !!! GET INFORMATION NECESSARY TO ACCESS FILES IN THE !!!          &
        !!! REMOTE NODE FOR OPANFW                              !!!       &
\       INPUT   "ENTER USER ID:";USERID$!GET USER ID                      &
\       INPUT   "ENTER PASSWORD:";PASSWD$!GET PASSWORD                    &
\       INPUT   "ENTER ACCOUNT NUMBER:";ACCNT$!GET ACCOUNT NUMBER         &
\       IDENT$=USERID$+NULL$+PASSWD$+NULL$+ACCNT$+NULL$                   &
\                                       !CREATE ASCIZ STRING FOR OPANFW

60      !!! CREATE NETWORK MAILBOX FOR ONLY ONE LINK !!!                  &
 \      CALL    OPNNTW BY REF (OPNLUN%,LSTAT%(),MSTAT%(),COUNT%)          &
 \      IF      LSTAT%(0%)=1% THEN 70   !IF OPEN NETWORK SUCCESSFUL,      &
                                        !OPEN REMOTE AND LOCAL FILES.     &
        ELSE    PRINT "MAILBOX CREATION ERROR" !IF UNSUCCESSFUL, PRINT    &

 \              PRINT "STATUS = ";LSTAT%(0%);",";LSTAT%(1%)               &
 \              GO TO 150               !MESSAGE AND STATUS AND EXIT

70      !!! OPEN LOCAL FILE !!!                                           &
 \      OPEN    BUF2$ FOR INPUT AS FILE #4

90      !!! OPEN REMOTE FILE FOR APPEND !!!                               &
 \      CALL    OPANFW BY REF(APPLUN%,ISTAT%(),NODNAM$,IDENT$,BUF1$,      &
                ICHAR$,LENGTH%,IBLOCK%)                                   &
 \      IF      ISTAT%(0%)=1% THEN 100  !IF SUCCESS, APPEND TO REMOTE     &
                                        !FILE FROM LOCAL FILE             &
        ELSE    PRINT "CAN NOT OPEN REMOTE FILE." !IF ERROR ON OPEN,      &
 \              PRINT "STATUS = ";ISTAT%(0%);",";ISTAT%(1%)               &
 \              GO TO 120               !OUTPUT MESSAGE AND STATUS AND    &
                                        !EXIT

100     !!!     READ RECORDS FROM LOCAL FILE AND WRITE THEM TO  !!!       &
 \      !!!     REMOTE FILE.                                    !!!       &
 \      FLAG% = 1%                      !SET FLAG FOR END OF FILE 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 100  !IF SUCCESSFUL, READ NEXT RECORD&
        ELSE    PRINT "WRITE ERROR FROM REMOTE FILE." !IF ERROR, PRINT    &
 \              PRINT "STATUS = ";JSTAT%(0%);",";JSTAT%(1%)               &
 \              GO TO 150               !MESSAGE AND STATUS AND EXIT

110     !!!  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 120  !IF SUCCESS DEACCESS NETWORK      &
        ELSE    PRINT " CAN NOT CLOSE REMOTE FILE." !IF CLOSE ERROR,      &
 \              PRINT "STATUS = ";KSTAT%(0%);",";KSTAT%(1%)               &
 \              GO TO 150               !PRINT MESSAGE, STATUS AND EXIT

120     CALL    CLSNT BY REF(NSTAT%())  !DEACCESS NETWORK                 &
 \      IF      NSTAT%(0%)=1% THEN 140  !IF SUCCESS, APPEND COMPLETE      &
        ELSE    PRINT "CAN NOT CLOSE NETWORK." !IF ERROR, PRINT           &
 \              PRINT "STATUS = ";NSTAT%(0%);",";NSTAT$(1%)               &
 \              GO TO 150               !MESSAGE AND STATUS AND EXIT
```

(continued on next page)

```
140     PRINT   "APPEND COMPLETE. END PROGRAM EXECUTION"                &
 \      GO TO   150

145     IF      ERR <> 11 THEN 146      !IF NOT EOF, PRINT ERROR        &
        ELSE    IF FLAG%=0% THEN 146    !IF EOF AND EOF FLAG NOT SET,   &
                                        !PRINT ERROR                    &
        ELSE    RESUME 110              !EOF SO CLOSE BOTH FILES

146     PRINT "ERROR ";ERR;" AT LINE ";ERL !PRINT ERROR AND LINE NO.

150     END
```

3.8.24 BASIC–PLUS–2 Remote File Access Programming Example (Read/Write)

The program BASRRW reads the contents of one remote file into another remote file. When an end-of-file is encountered, the last record is written to the remote file and both files are closed.

NOTE

This programming example is also included in your tape or disk kit.

```
10      !!!    To task build you must edit the task build command   !!! &
        !!!    file and the ODL file created by the build.          !!! &

        !!!     >Add the lines                                      !!! &
        !!!                                                         !!! &
        !!!             ACTFIL=4                                    !!! &
        !!!             EXTSCT=$$FSR1:10000                         !!! &
        !!!     to the task build command file.                     !!! &
        !!!                                                         !!! &
        !!!     >Append                                             !!! &
        !!!                                                         !!! &
        !!!             -NETLIB-NETLB2                              !!! &
        !!!     to the USER: line of the ODL file.                  !!! &
        !!!                                                         !!! &
        !!!     >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 &
 \      INPLUN%=1%                      !LUN ASSIGNED TO INPUT FILE     &
 \      OUTLUN%=2%                      !LUN ASSIGNED TO OUTPUT FILE    &
 \      EOF%=20519%                     !END OF FILE STATUS RETURN      &
 \      FALSE%=0%                       !FLAG INDICATING FALSE          &
 \      TRUE%=-1%                       !FLAG INDICATING TRUE

40      !!! INPUT NODE NAME !!!                                         &
        INPUT   "ENTER REMOTE NODE NAME (MAX. 6 CHARACTERS):";NODNAM$   &
 \      IF      LEN(NODNAM$)>6% THEN PRINT                              &
                "NAME TOO LONG, PLEASE RE-ENTER"                        &
 \              PRINT \ GO TO 40                                        &
        ELSE    NODNAM$=NODNAM$+NULL$   !CREATE ASCIZ STRING

50      !!! GET INFORMATION NECESSARY TO ACCESS FILES IN THE  !!!       &
        !!! REMOTE NODE.                                      !!!       &
 \      INPUT   "ENTER USER ID:";USERID$!GET USER ID                    &
 \      INPUT   "ENTER PASSWORD:";PASSWD$!GET PASSWORD                  &
 \      INPUT   "ENTER ACCOUNT NUMBER:";ACCNT$!GET ACCOUNT NUMBER       &
 \      IDINFO$=USERID$+NULL$+PASSWD$+NULL$+ACCNT$+NULL$                &
 \                                      !CREATE ASCIZ STRING
```

(continued on next page)

```
60      !!! INPUT FILE SPECIFICATION FOR INPUT & OUTPUT FILES !!!       &
        INPUT   "INPUT FILE SPEC. FOR INPUT FILE";INFIL$                &
 \      INFIL$=INFIL$+NULL$             !APPEND NULL CHARACTER          &
 \      INPUT   "INPUT FILE SPEC. FOR OUTPUT FILE";OUTFIL$              &
 \      OUTFIL$=OUTFIL$+NULL$           !APPEND NULL CHARACTER

70      !!! DECLARE NETWORK TASK !!!                                    &
        CALL    OPNNTW BY REF(OPNLUN%,ISTAT%(),MSTAT%(),COUNT%)         &
                                        !ACCESS NETWORK                 &
 \      LOC1=1                          !ORIGIN OF CALL FOR SUBROUTINE  &
 \      GOSUB   150                     !CHECK STATUS

80      !!! OPEN FILE FOR INPUT. THE FILE RESIDES ON A REMOTE NODE !!!  &
        CALL    OPRNFW BY REF(INPLUN%,ISTAT%(),NODNAM$,IDINFO$,INFIL$,  &
                ICHARS$,LNTH%,BLOCK%)   !OPEN REMOTE FILE FOR INPUT     &
 \      LOC1=2                          !ORIGIN OF CALL FOR SUBROUTINE  &
 \      GOSUB   150                     !CHECK STATUS

90      !!! OPEN FILE FOR OUTPUT. FILE RESIDES ON A REMOTE NODE !!!     &
        CALL    OPWNFW BY REF(OUTLUN%,ISTAT%(),NODNAM$,IDINFO$,OUTFIL$, &
                ICHARS$,LNTH%,BLOCK%)   !OPEN REMOTE FILE FOR OUTPUT    &
 \      LOC1=3                          !ORIGIN OF CALL FOR SUBROUTINE  &
 \      GOSUB   150                     !CHECK STATUS

100     !!! ONCE THE FILES ARE SUCCESSFULLY OPENED, WE MAY      !!!     &
        !!! TRANSFER RECORDS.  THE FILE ASSOCIATED WITH LUN 1   !!!     &
        !!! IS OPENED FOR READING, THE FILE ON LUN 2 IS OPENED  !!!     &
        !!!                     FOR WRITING.                    !!!     &
        FOR I%=1% TO 100%

102             CALL    GETNFW BY REF(INPLUN%,ISTAT%(),256%,IARRAY%())  &
                                        !READ A RECORD FROM INPUT FILE

104             IF ISTAT%(0%)<>1% AND ISTAT%(1%)<>EOF% THEN 110         &
                                        !IF UNSUCCESSFUL, PRINT MESSAGE &
                ELSE LNTH%=ISTAT%(1%)   !SAVE NO. OF BYTES TRANSFERRED

105     !!! CHECK FOR END OF FILE !!!                                   &
                IF ISTAT%(0%)<>1% AND ISTAT%(1%)=EOF% THEN EOFFG%=TRUE% &
                ELSE EOFFG%=FALSE%      !SET FLAG IF END OF FILE

106             IF EOFFG%=TRUE% THEN 130!IF END OF FILE, CLOSE FILES    &
                ELSE    CALL PUTNFW BY REF(OUTLUN%,ISTAT%(),LNTH%       &
                        ,IARRAY%())     !WRITE RECORD

107             IF ISTAT%(0%)<>1% AND ISTAT(1%)<>EOF% GO TO 120         &
                ELSE A%=1%              !IF UNSUCCESSFUL, PRINT MESSAGE
108     NEXT I%                         !TERMINATE LOOP

110     !!! READ ERROR OCCURRED !!!                                     &
        PRINT   "READ ERROR. STATUS = ";ISTAT%(0%);",";ISTAT%(1%)       &
 \      GO TO   130                     !CLOSE BOTH FILES

120     !!! WRITE ERROR OCCURRED !!!                                    &
        PRINT   "WRITE ERROR. STATUS = ";ISTAT%(0%);",";ISTAT%(1%)

130     !!! 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 140  !IF FLAG NOT TRUE, TRANSFER NOT &
                                        !SUCCESSFUL                     &
        ELSE    PRINT "END OF FILE REACHED. FILE CLOSED"                &
                                        !INDICATE TRANSFER SUCCESSFUL
```

(continued on next page)

```
140     GOTO    170                     !BRANCH TO END

150     !!! SUBROUTINE TO CHECK STATUS ON COMPLETION OF OPEN CALLS !!!  &
        IF      ISTAT%(0%)=1% THEN 160  !IF SUCCESS, JUST RETURN        &
        ELSE    PRINT "OPEN ERROR. STATUS = ";ISTAT%(0%);",";ISTAT%(1%) &


 \              PRINT "LOC = ";LOC1     !ORIGIN OF CALL                 &
 \              GO TO 170               !QUIT IF UNSUCCESSFUL

160     RETURN                          !EXIT FROM SUBROUTINE

170     END                             !END EXECUTION
```

3.9 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.

Table 3–4: FORTRAN Task Control Call Summary

Call	Function
ABONCW	Abort an executing task or cancel a scheduled task
BACUSR	Build accounting information and the user ID portion of the connect block
RUNNCW	Execute an installed task in a remote node

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 deaccess the network.

3.9.1 Waiting for Requests

All calls are synchronous and pass control back to the user task only after the operation completes.

3.9.2 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 will complete with an error.

3.9.3 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*,*indnm*,*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) will contain the disconnect or reject reason code. Refer to Appendix A. Otherwise, *status*(2) will contain 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 node to which this abort request is directed.

passwdsiz

specifies an integer variable or constant containing the password length in bytes.

passwd

specifies a 1- to 8.-element byte array containing the user password (the password you use to log on to the remote system) for the node you want to access.

A privileged password allows a user to abort any task running on the remote node without specifying the *ident* parameter. A nonprivileged password allows a user to abort a task if the correct *ident* parameter is specified.

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 you want 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 the way ABONCW is used. This is an optional argument. If you omit this argument or if its value equals 0, only the executing task you specified in the call is aborted.

If the value equals 1, the rescheduling of the specified task is cancelled and the presently active task continues to execute.

If the value is greater than 1, the executing task is aborted and rescheduling of the task is cancelled.

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.*

3.9.4 BACUSR – Build Account and User ID Information Area

Use:

Call BACUSR in the task requesting the connection if access control information (user ID and account) is not passed to the remote node in an alias node name specification. BACUSR allows you to build the account and user ID areas of the connect block for task control programming.

BACUSR is similar in function to BACC. Unlike BACC, however, BACUSR does not allow you to specify a password. The password is specified in a task control program as a parameter of a call (ABONCW or RUNNCW).

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, or 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. *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 (not used for RSX target systems).

accno

specifies a 1- to 16.-byte array containing the account number. *Accnosz* and *accno* are paired optional arguments. Include both or omit both.

3.9.5 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) will contain the disconnect or reject reason code. Refer to Appendix A. Otherwise, *status*(2) will contain a directive error or will not be 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 node to which this request is directed.

passwdsiz

specifies an integer variable or constant containing the password length in bytes.

passwd

specifies a 1- to 8.-element byte array containing the user password (the password you use to log on to the remote system) for the node you want to access.

A privileged password allows you to run a task under any user identification code on the remote node. A nonprivileged password allows you to run a task only under 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 you want to execute.

* *ident*

specifies an integer variable containing the negated task control block address of the remote task when RUNNCW completes. This value will be used in ABONCW. If you do not plan to cancel or abort this task later on, you do not need to include this argument.

uic

specifies a 2-byte array containing the group and user codes under which the task will run 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 will run 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 be 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. Time unit codes and their meanings are:

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 be used 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 you want the task to run for the first time in one hour and then run 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 that is 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.*

3.9.6 FORTRAN Task Control Programming Example

The program RUNABO.FTN uses DECnet task control calls to run or abort a task on a specified local or remote node. After your task control request has been executed, you will be prompted to enter another request to run or abort the associated task. When you have finished entering the task control requests, press CTRL/Z to exit from the request-prompting loop and stop the program.

Before running RUNABO.FTN, the TCL task must be installed on the target node as shown at the beginning of the program.

NOTE

This programming example is also included in your tape or disk kit.

```
C
C  RUNABO.FTN
C
C       - USES THE DECNET TASK CONTROL CALLS TO RUN OR ABORT A
C           TASK ON THE SPECIFIED NODE (LOCAL OR REMOTE).
C       - TASK "TCL"  MUST BE INSTALLED AT THE TARGET NODE.s
C       - SAMPLE COMMANDS TO DEVELOP THIS TASK FOLLOW
C
C               >FOR RUNABO,RUNABO=RUNABO
C               >TKB RUNABO,RUNABO=RUNABO,[1,1]NETFOR/LB
C               >INS RUNABO/TASK=...CTL
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)
        DATA RUN/'R'/,ABO/'A'/
C
C  CREATE 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,$'TARGET NODE ? :')
        READ(5,2,END=999)ICNT1,TARNOD
   2    FORMAT(Q,16A1)
C
        WRITE(5,3)
   3    FORMAT(5X,$'TARGET TASK ? :')
        READ(5,2,END=999)ICNT2,TARTSK
C
C PROMPT FOR ACCESS CONTROL INFORMATION
C
        WRITE(5,50)
  50    FORMAT(5X,$'TARGET USER I.D. ?:')
        READ (5,2,END=999)ICNT4,USERID
C
        WRITE(5,4)
   4    FORMAT(5X,$'TARGET PASSWORD ?:')
        READ (5,2,END=999)ICNT3,PASSWD
```

(continued on next page)

```
C
        WRITE(5,11)
  11    FORMAT(5X,$'TARGET ACCOUNT NUMBER ?:')
        READ (5,2,END=999)ICNT5,ACCNT
C
        WRITE(5,6)
   6    FORMAT(5X,$'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.
  20    WRITE (5,100)
 100    FORMAT (5X,$'IF A PRIVILEGED PASSWORD IS USED, ENTER 0. IDENT ?:')
        READ (5,110)IDENT
 110    FORMAT(I6)
        CALL ABONCW (2,STATUS,ICNT1,TARNOD,ICNT3,PASSWD,ICNT2,TARTSK,IDENT)
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: Direct Line Access Controller

DLX allows programs to send and receive data that bypasses the standard DECnet user interface. This permits:

- Communication with non-DECnet based systems
- Reduction of overhead messages exchanged with other systems or systems that may be running DLX

To use DLX, you issue queued input/output (QIO) calls to the NX: device. The DLX interface can be used to communicate over all devices supported by DECnet–RSX.

DLX is automatically built for RSX–11M–PLUS systems. It is optional for RSX–11M/RSX–11S systems. However, DLX is required for down-line loads and up-line dumps from RSX–11S systems.

DLX programming requires a thorough knowledge of MACRO–11 assembly language and experience in writing real-time application programs. You must write tasks that synchronize with each other before transferring data. If tasks are not synchronized, the data can be lost during task-to-task communication. You must provide your own error-handling routines. The DLX software informs your task of any errors, but your task must be written to process error recovery.

You can use DLX QIOs to communicate between your program and a program on an adjacent node using the DECnet DDCMP protocol or the Ethernet. The adjacent system can be any DECnet–RSX or non-DECnet based system that has similar capabilities. In task-to-task communication between adjacent nodes, DLX significantly improves network performance in terms of CPU and line usage. You can build your own user level protocol that best suits the application.

NOTE

All messages transmitted and received via DLX are buffered in network buffers. In previous versions of DECnet–RSX, user programs needed to define buffer space for user data as well as protocol overhead using the DLXBUF macro. This is no longer required.

4.1 System Requirements for Tasks Using DLX

Running programs which use the DLX interface requires some special considerations.

- The DLX process must be loaded. DLX is loaded in the system as a common area called NT.DLX. It will normally be resident after loading the network with an NCP or VNP command (for example, NCP SET SYSTEM or VNP SET SYSTEM command).
- Prior to running the program, it is necessary to set the line states for the devices the program will use. The details for setting lines are described for Ethernet and non-Ethernet devices.

If the program will use a non-Ethernet device, it is necessary to use the following commands. The variable *dev-x* identifies a specific line (for example, DMC–0).

>NCP SET LINE *dev-x*
>NCP SET LINE *dev-x* OWNER DLX
>NCP SET LINE *dev-x* STATE ON

Now you are ready to run the program.

If the program will use an Ethernet device, it is necessary to use the following command. The variable *dev-x* identifies a UNA or QNA device (for example, UNA–1).

>NCP SET LIN *dev-x*

Now you are ready to run the program.

4.2 Special Considerations for Ethernet Users

The Ethernet is unlike other data links supported by the Communications Executive (CEX) and DECnet in that a single circuit can be used by more than one user simultaneously. Externally, Ethernet devices appear to be single line point-to-point controllers (for example, UNA–0, UNA–1). Internally, they are implemented as multipoint devices, with each station representing an available port onto the Ethernet.

Because the Ethernet allows multiple users to access the physical link simultaneously, some mechanism must be provided to deliver received messages to the correct user. All messages on the Ethernet must include a destination address (48-bit) and a protocol type (16-bit). There are two modes that determine how messages will be transmitted: physical address mode and multicast address mode.

Physical address mode defines a unique address for a single node on any Ethernet. Multicast address mode defines a multidestination address of one or more nodes on a given Ethernet. With multicast addressing, any number of nodes can be assigned a group address so they are all able to receive the same data in a single transmission. Before transmitting and receiving messages, you must define a specific mode. To do this, use the SET CHARACTERISTICS (IO.XSC) QIO call (Section 4.3.2).

Each user must enable unique protocol/address pairs to define which messages it wishes to receive. For example, user 1 may enable protocol A to addresses 1 and 2, while user 2 may enable protocol B to addresses 3 and 4. It is possible for two or more users to enable the same protocol or addresses, providing that the protocol/address pairs are unique.

The Ethernet may be opened in three different modes (defined in EPMDF$ in NETLIB.MLB):

- Exclusive LF$EXC – This user has exclusive use of the specific protocol and no other user may transmit or receive using this protocol. (DECnet routing uses this mode.)
- Default LF$DEF – This user should receive messages on this protocol that would otherwise be discarded because there was no protocol/address pair set up.
- Normal – The user must specify the protocol/address pairs that will be used for communications.

In addition, the user can select padding for an Ethernet (LF$PAD) that will prefix the message with a 2-byte length field and pad the message out to the minimum Ethernet size on transmit. On receive, the length field will be used to indicate the amount of data present.

When a hardware error is detected on the Ethernet controller, all protocol/address pairings and multicast addresses will be lost. After issuing the IO.XIN call to reinitialize the channel, the protocol/address pairs and the multicast addresses must be reenabled.

Ethernet users should refer to the *DECnet–RSX Network Concepts and Procedures* manual for more information.

4.3 DLX QIOs

DLX requests conform to normal RSX–11 QIO standards. Standards for logical unit numbers (LUNs), event flags, I/O status blocks, asynchronous system traps (ASTs), and parameter lists are observed. According to RSX–11 standards, any one of the three macro formats may be used (see Section 2.1). The QIO wait option (specified as QIOW$) also may be used to suspend further execution of the program until the call completes.

The macros are defined in the DECnet macro library (NETLIB.MLB). This library is transferred to the user's system during NETGEN. The definitions and offsets used in the macros are contained in two definition macros: DLXDF$ and EPMDF$.

It is necessary to issue .MCALL statements and explicitly invoke the macro in the user program. For example,

```
.MCALL  DLXDF$,EPMDF$
          .
          .
          .
        DLXDF$
        EPMDF$
```

The DLX QIO codes and functions are summarized in Table 4–1 for both Ethernet and non-Ethernet. Each call, with its arguments and completion status codes, is described in this chapter (Sections 4.3.1 through 4.3.7).

Table 4–1: Summary of DLX Function Codes

Code	Non-Ethernet	Ethernet
IO.XOP	Open a line	Open the Ethernet device
IO.XSC	Not Applicable	Set characteristics
IO.XIN	Initialize the line	Not Applicable
IO.XTM	Transmit a message	Transmit a message
IO.XRC	Receive a message	Receive a message
IO.XHG	Hang up the line	Not Applicable
IO.XCL	Close the line	Close the line

4.3.1 IO.XOP – Open a Line

Use:

Issue this QIO to open a line for direct line access message transfer and reception. This QIO causes the specified LUN to be associated with the specified line. The line is then implicitly initiated, and the protocol is started. The line owner must be DLX; the line must be either ON or in SERVICE state, and the LUN must have been assigned to NX:. Note that in normal mode (see the *p3* argument, below), this function will not complete until the task at the other end of the line also performs an open or initialize function.

To open the Ethernet device from DLX, specify the address in argument *p*1 for the device ID string (for example, UNA–0). DLX will scan the port database for an available port and assign it to DLX for the user.

Format:

QIO$ IO.XOP,*lun*,[*efn*],,[*status*],[*ast*],<*p*1,*p*2,*p*3>

Arguments:

IO.XOP

is the function code that opens a line.

lun

is the logical unit number associated with the line that you are opening.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

p1

is the address of an ASCII string that identifies the line to be opened.

For non-Ethernet devices the format is:

dev-ctl[*-line*][*.tributary*]

where *dev* is the device mnemonic, *ctl* is the decimal value for the controller number, *line* is the decimal number of the line you are opening, and *tributary* defines the decimal number of the multipoint tributary with which you want to communicate.

p2

is the length of the line identification field.

For Ethernet devices, the format is:

dev-ctl

where *dev* is UNA or QNA, and *ctl* is the decimal value for the controller number.

p3

is a word argument that specifies the line mode and timeout value for the call. (The timeout value is the amount of time that the receiver waits for a message to be transmitted.) 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*.) The high-order byte of this word designates the line mode as follows:

mode = 0 for normal mode.
mode = 1 for Maintenance Operation Protocol (MOP) mode.

Completion Status:

IS.SUC (1)	The line has been 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 have entered an invalid line identification format or the specified line is not in the system.
177760 IE.PRI (-16.)	The specified line is not available for use by DLX.
177757 IE.RSU (-17.)	The specified line is already in use.

4.3.2 IO.XSC – Set Characteristics (Ethernet Only)

Use:

Use this Ethernet QIO to set up the protocol/address pairs and multicast addresses.

Format:

QIO$ IO.XSC,*lun*,[*efn*],,[*status*],[*ast*],<*p1*,*p2*>

Arguments:

IO.XSC

is the function code that supplies a single characteristics buffer in arguments *p1* and *p2*. This buffer may contain multiple characteristics blocks.

lun

is the logical unit number associated with the line that you are setting for a characteristics buffer.

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 will indicate how much of the characteristics buffer has been processed.

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

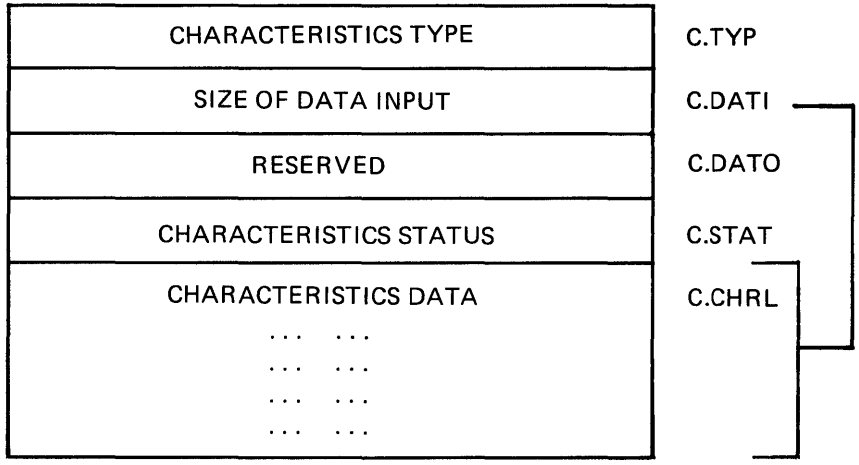
p1

is the address of the characteristics buffer.

p2

is the length of the characteristics buffer.

The set characteristics buffer format may contain multiple characteristics blocks. Each characteristics block has the following format:



Common error codes in C.STAT are:

CE.UDF Undefined function

CE.RTS Request too small (not enough data supplied)

CE.RTL Request toc large (too much data supplied)

CE.RES Resource allocation failure

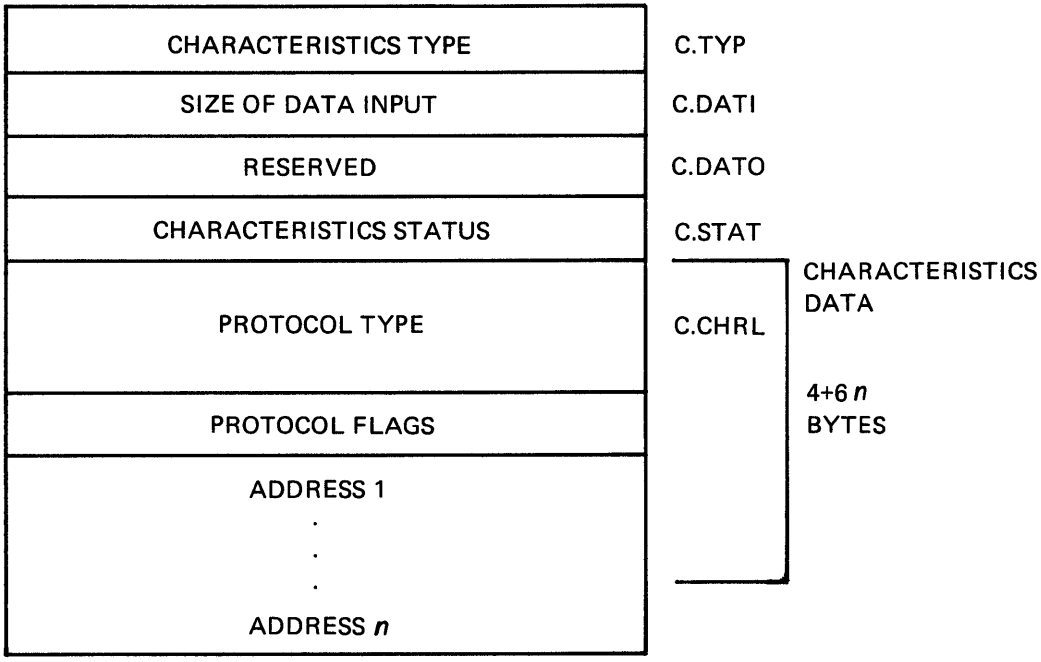
The size of data input (C.DATI) indicates how many bytes of characteristic data are being supplied, the size of data output (C.DATO) is not used by this function. The status field (C.STAT) of those characteristics blocks that have been processed will be set to indicate the success or failure of the characteristics function. Protocol flags are defined in EPMDF$ (LF$*xxx*).

NOTE

The address field(s) should not be present if LF$EXC or LF$DEF is specified in the flags.

Setting up protocol/address pairs:

Protocol type = CC.DST (200) — allows transmission and reception of messages with the specified protocol to/from any of the addresses in the list.



Errors returned in C.STAT are:

CE.PCN Protocol usage conflict:

A. Another user has exclusive access to this protocol.

B. There is already a default user of this protocol and this request is attempting to set up a new default user.

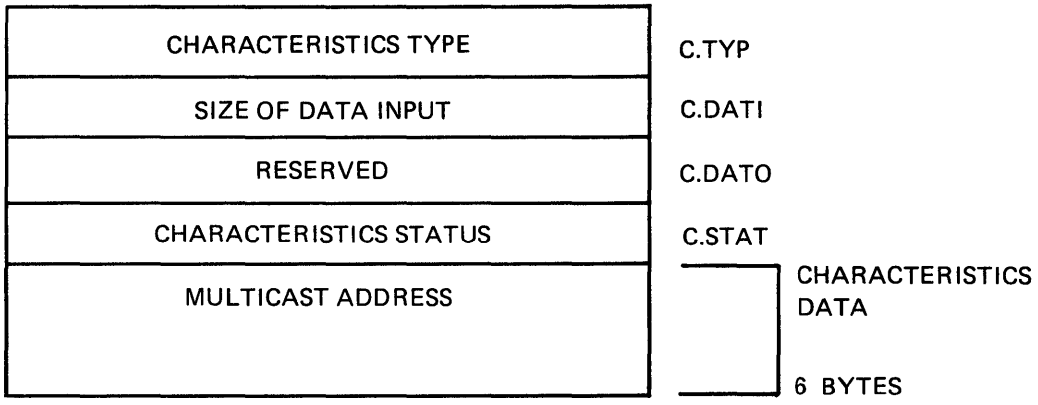
C. The padding status of this protocol does not match that which is requested.

CE.IUN Illegal use of multicast address; one of the addresses specified is multicast.

CE.ACN Address usage conflicts; the protocol/address pair is already in use.

Setting up a multicast address:

Protocol type = CC.MCT (201) — allows reception of messages that are sent to the specified multicast address.



Errors returned in C.STAT are:

CE.NMA Not a multicast address.

CE.MCE Multicast address already enabled.

4.3.3 IO.XIN – Initialize the Line (non-Ethernet only)

Use:

Issue this QIO to reinitialize a line after a fatal device error has occurred. When you use this QIO, you must reset the mode and timer values. Note that in normal mode (see the *p1* argument, below), this function will not complete until the task at the other end of the line 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 line.

lun

is the logical unit number associated with the line that you are initializing.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

p1

is both the mode and timer argument. Use the same format as described for argument *p3* in IO.XOP (Section 4.3.1).

Completion Status:

IS.SUC (1)	The line has been successfully initialized.
177761 IE.ABO (-15.)	The initialization attempt has been aborted. This could have been caused by a hardware device error, a user-issued hang-up QIO, or an attempt to initialize a line that was not hung up.
177776 IE.IFC (-2.)	The LUN is not assigned to NX.
177733 IE.NLN (-37.)	No line has been opened with the specified LUN.

4.3.4 IO.XTM – Transmit a Message on the Line

Non-Ethernet Use:

Issue this QIO to transmit a message on a line that has been initialized (see Section 4.3.3). The data that you transmit to the adjacent node is transferred from your user buffer and copied to a network buffer for transmission.

Ethernet Use:

When transmitting a message on the Ethernet, you must specify the destination address for the multicast address to be used for this message along with the protocol type. This is accomplished by having an optional auxiliary characteristics buffer for transmit in arguments *p3* and *p4*.

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 for the line on which you are transmitting data.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

p1

is the address of the user buffer that contains the message to be transmitted. Use the label specified in the DLXBUF macro call.

p2

is the length of the message you are sending to the remote node (excluding the DDCMP header and checksum).

p3

is the address of the optional auxiliary characteristics buffer multicast addresses.

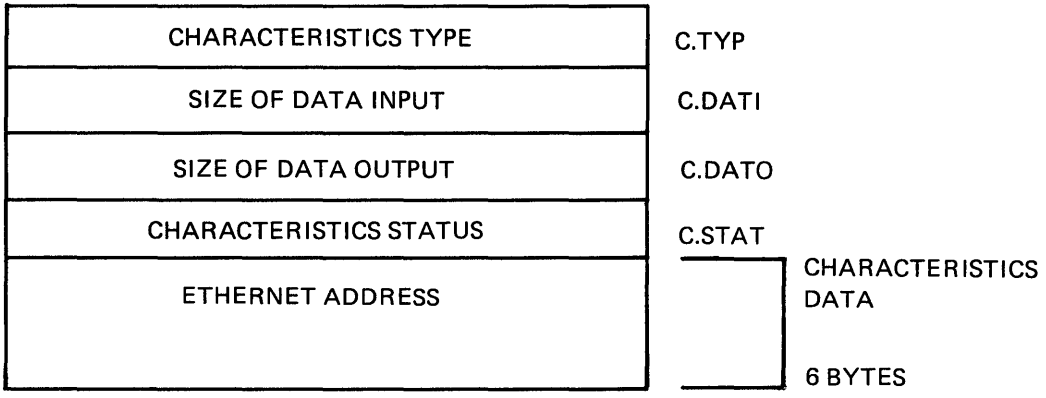
p4

is the length of the optional auxiliary characteristics buffer.

The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 4.3.2. The individual characteristics block has the following format:

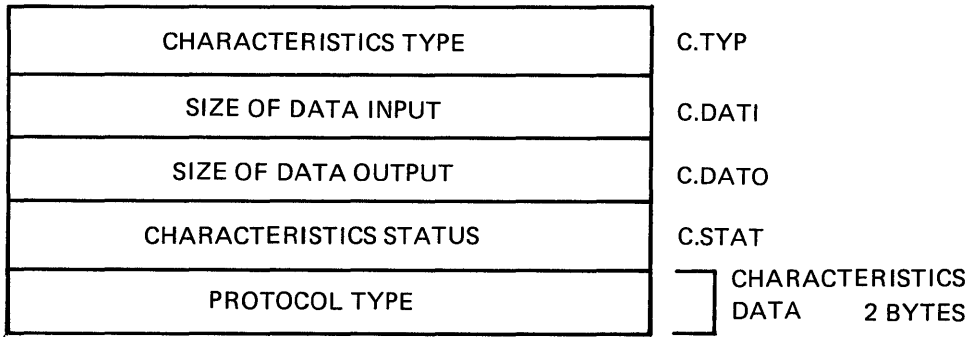
To set the Ethernet address:

Protocol type = CC.ADR (100)



To set the protocol type:

Protocol type = CC.PRO (101)



Transmit requests on Ethernet channels must include an auxiliary characteristics buffer including both the address and protocol type. Failure to do so will cause the transmit message to be returned with a IE.BAD error. If the auxiliary buffer is present for other data links, the individual characteristics blocks will be completed with a CS.IGN (Successful, block ignored) error in C.STAT.

Completion Status:

IS.SUC (1)	The message was transmitted to the remote node successfully.
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 line is hung up. You must either issue a QIO to initialize the line (see Section 4.3.3) or close and reopen the line (see Sections 4.3.7 and 4.3.1, respectively) before you can use that line again.
177775 IE.DNR (-3.)	The hardware device was not ready. The line was hung up and has not been reinitialized.
177776 IE.IFC (-2.)	The LUN is not assigned to NX.
177733 IE.NLN (-37.)	No line has been opened with the specified LUN.
177772 IE.SPC (-6.)	The transmit buffer is too large (applicable only to PDP-11/44 or PDP-11/70 with extended memory).

4.3.5 IO.XRC – Receive a Message on the Line

Non-Ethernet Use:

Issue this QIO to receive a message from the remote node on a line that has been initialized (see Section 4.3.3). Unless you issue a receive QIO, any data sent to you by a remote node is lost. When you open a line in MOP (Maintenance Operation Protocol) mode, this loss of data is not reported to you. If, however, you open the line in normal mode, an error is reported to you the next time you issue a receive QIO.

Ethernet Use:

When receiving a message on the Ethernet, you must find out the source address for this message along with the protocol type. This is accomplished by having an optional auxiliary characteristics buffer for receive messages in parameters *p*3, and *p*4.

Format:

QIO$ IO.XRC,*lun*,[*efn*],,[*status*],[*ast*],<*p*1,*p*2,[*p*3,*p*4]

Arguments:

IO.XRC

is the function code for receiving a message.

lun

is the logical unit number associated with the line on which you 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 that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

p1

is the address of the user buffer in your system that receives the message.

p2

is the length in bytes that you are allocating for the receive buffer. (The received message cannot be longer than the size of the system buffer, regardless of the length you state here for *p2*.)

p3

is the address of the optional auxiliary characteristics buffer.

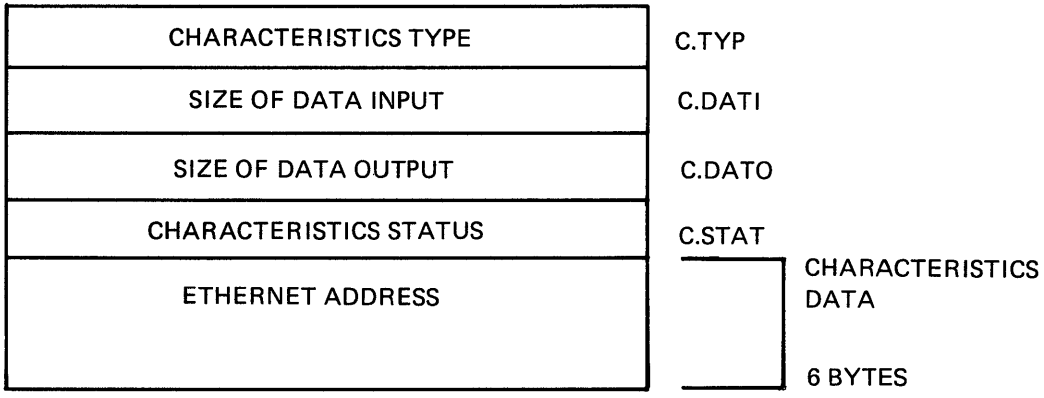
p4

is the length of the optional auxiliary characteristics buffer.

The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 4.3.2. The individual characteristics block has the following format:

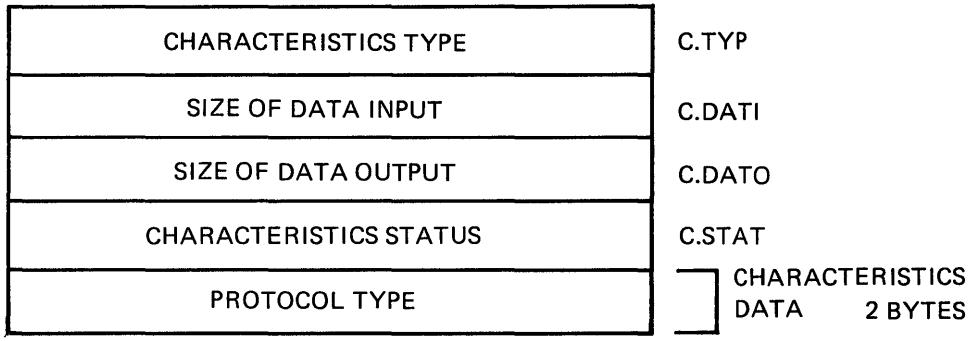
To read the Ethernet address:

Protocol type = CC.ADR (100)



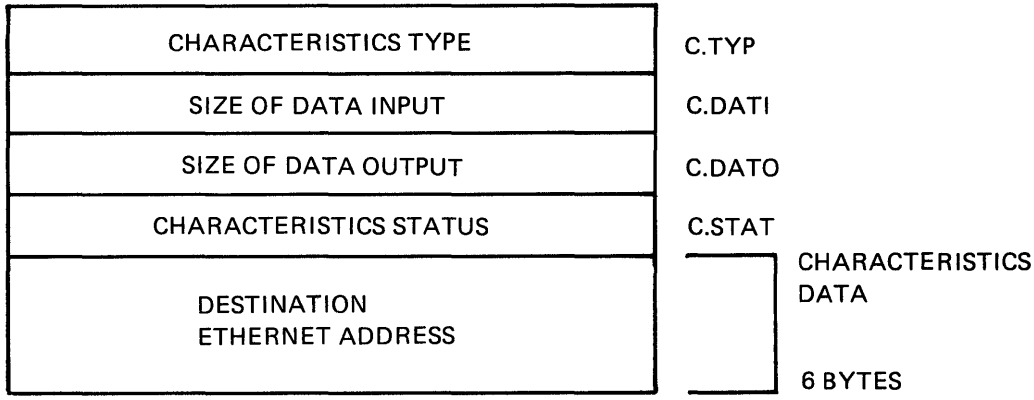
To read the protocol type:

Protocol type = CC.PRO (101)



To read destination Ethernet address:

Protocol type = CC.DAD (102)



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 actually received.
177761 IE.ABO (-15.)	The receive function 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 receive is aborted, the line is hung up. You must either issue an initialize QIO (see Section 4.3.3) or close and reopen the line (see Sections 4.3.7 and 4.3.1, respectively) before you can use the line again.
177763 IE.DAO (-13.)	Either a message was received before a receive QIO was issued and the data is lost (this applies only to normal mode operations), or the user buffer was too small to receive all of the data. In the latter case, the message is truncated, and some data is lost. (The length of the user buffer is contained in the second word of the I/O status block.)
177775 IE.DNR (-3.)	The hardware device was not ready. The line was hung up and has not been reinitialized.
177776 IE.IFC (-2.)	The LUN is not assigned to NX.
177733 IE.NLN (-37.)	No line has been opened with the specified logical unit number.
177641 IE.TMO (-95.)	A timeout condition has occurred. No message was received within the timer interval specified when you opened or initialized the line.
177774 IE.VER (-4.)	An error has occurred on the line. The second word of the I/O status block contains the error code. Possible error codes and their meanings are: 100361 DDCMP transmit error threshold exceeded 100362 Operation aborted 100363 Message received without receive pending 100364 Start received 100366 Line physically disconnected 100370 General error 100372 MOP message received 100374 DDCMP reply timeout threshold exceeded 100376 DDCMP receive error threshold exceeded

4.3.6 IO.XHG – Hang Up the Line (non-Ethernet only)

Use:

Issue this QIO to stop operations on a line. This QIO does not close a line. However, you must issue an initialize QIO (see Section 4.3.3) or close and reopen QIOs (see Sections 4.3.7 and 4.3.1, respectively) to resume operations.

Format:

QIO$ IO.XHG,*lun*,[*efn*],,[*status*],[*ast*]

Arguments:

IO.XHG

is the function code that hangs up the line.

lun

is the logical unit number associated with the line you are hanging up.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

Completion Status:

IS.SUC (1)	This line was hung up successfully.
177776 IE.IFC (-2.)	The LUN is not assigned to NX.
177733 IE.NLN (-37.)	No line has been opened with the specified LUN.

4.3.7 IO.XCL – Close the Line (non-Ethernet only)

Use:

Issue the IO.XCL call to close an open line and stop the protocol. If you have a dial-up connection, however, the line will not be hung up before it closes.

Format:

QIO$ IO.XCL,*lun*,[*efn*],,[*status*],[*ast*]

Arguments:

IO.XCL

is the function code that closes the line.

lun

is the logical unit number associated with the line that you are closing.

efn

is an optional event flag number set when the call completes.

status

is the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status, below).

ast

is the entry point into an optional user-written AST routine to be executed after this call completes.

Completion Status:

IS.SUC (1)	The line has been successfully closed.
177776 IE.IFC (–2.)	The LUN is not assigned to NX.
177733 IE.NLN (–37.)	No line has been opened with the specified LUN.

4.3.8 DLX QIO Transmit Programming Example (for non-Ethernet device)

The following programs are examples of tasks using the DECnet–RSX DLX interface. XTS–DLX TRANSMITTER uses DLX QIOs to send data. XTR–DLX RECEIVER uses DLX QIOs to receive messages.

NOTE

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

```
        .TITLE  XTS - DLX TRANSMITTER
        .IDENT  /V01.01/

;
; COPYRIGHT (C) 1983, 1985 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.
;
;
; MODULE DESCRIPTION:
;
;       XTS - DLX TRANSMITTER
;
;
; DISTRIBUTED SYSTEMS SOFTWARE ENGINEERING
;
; IDENT HISTORY:
;
; 1.00  28-OCT-83
;       VERSION 1.0 RELEASE
;
; 1.01  11-MAR-85
;       Correct build procedure UICs
;
```

(continued on next page)

```
;+
;
; XTS - DLX SYSTEM EXERCISER (THIS TEST UTILITY IS UNSUPPORTED)
;
; XTS IS A UITILITY WHICH ENABLES A USER TO TRANSMIT DATA READ IN FROM
; A TERMINAL OR COMMAND FILE ACROSS AN "ERROR FREE"
; LINE TO A RECEIVER TASK WHICH ECHOES THE RECEIVED DATA BACK OVER THE
; CHANNEL. XTS USES THE DLX INTERFACE TO PROVIDE THE "ERROR FREE" CHANNEL.
;
; BREAK THROUGH WRITES MUST BE SUPPORTED IN ORDER TO RUN THIS PROGRAM.
;
;
;    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,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 (IF ANY) HAS BEEN PERFORMED.
;
;
; THE FOLLOWING IS AN EXAMPLE OF THE XTS 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
;       >
;
; IN ORDER FOR XTS TO RECEIVE AN ECHO OF THE MESSAGE, XTR MUST BE RUNNING.
; IT IS INITIATED IN THE FOLLOWING MANNER:
;
;       >XTR
;       LINE: DUP-0
;
; WHEN FINISHED WITH XTS/XTR, XTR MUST BE ABORTED
;-

        .SBTTL  LOCAL MACROS

        .MACRO  EPRINT  ERRMSG
        MOV     #ERRMSG,R0
        CALL    $EPRINT
        .ENDM   EPRINT
```

(continued on next page)

```
.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 CHANNEL
        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 CHANNEL
        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)

;****
; DPB'S
;****

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
```

(continued on next page)

```
;
; CHANNEL I/O STATUS BLOCK
;
CHNSB:  .BLKW   2

;
; AST SAVED I/O STATUS BLOCK
;
IOSB:   .BLKW   1

;
; CHANNEL 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

;
; CHANNEL 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 -- /
BUFREC: .BLKB   7
        .LIST   BEX
        .EVEN
```

(continued on next page)

```
        .SBTTL  XTS - XTS MAIN LINE
;+
; XTS -- MAIN LINE OF XTS CODE
;-

XTSEP::

        MOV     #EX$SUC,EXSTAT          ;ASSUME EXIT WITH STATUS
;
; ASSIGN LUN TO CHANNEL
;
        ALUN$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
;
```

(continued on next page)

```
; 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 IS CALLED TO GET A COMMAND LINE FOR XTS. INPUT CAN BE
; FROM TI: OR AN INDIRECT COMMAND FILE. RETURN WITH C-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 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,  RE-PROMPT
30$:    SEC                             ;SET-C
        BR      50$                     ;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                          ;RETURN
```

(continued on next page)

```
        .SBTTL  HNGRC1 - HANG ASYNCHRONOUS READ ON LINE
;+
; **-HNGRC1 - HANG AN ASYNCHRONOUS READ ON THE CHANNEL
; **-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
```

(continued on next page)

```
        .SBTTL  RECAST - AST FOR CHANNEL READ COMPLETE
;+
; **-RECAST - AST FOR CHANNEL 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
```

(continued on next page)

```
        .SBTTL  ECHO - ECHO COMMAND LINE
;+
; **-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 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
```

4.3.9 DLX QIO Receiver Programming Example (for non-Ethernet device)

The DLXRCV program uses DLX QIOs to receive data for Ethernet devices only.

NOTE

This programming example is also included in your tape or disk kit.

```
        .TITLE  XTR - DLX RECEIVER
        .IDENT  /V01.01/

;
; COPYRIGHT (C) 1983, 1985 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.
;
;
; MODULE DESCRIPTION:
;
;       XTR - DLX RECEIVER
;
;
; DISTRIBUTED SYSTEMS SOFTWARE ENGINEERING
;
; IDENT HISTORY:
;
; 1.00  28-JUL-79
;       VERSION 1.0 RELEASE
;
; 1.01  11-MAR-85
;       Correct build procedure UICs
;
```

(continued on next page)

```
;+
;
; XTR - DLX SYSTEM EXERCISER (THIS TEST UTILITY IS UNSUPPORTED)
;
; XTR ECHOES RECEIVED DATA BACK OVER THE CHANNEL. XTR USES THE DLX UTILITY
; TO PROVIDE THE "ERROR FREE" CHANNEL.
;
;
;    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,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 (IF ANY) HAS BEEN PERFORMED.
;
;
; THE FOLLOWING IS AN EXAMPLE OF THE XTS 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
;       >
;
; IN ORDER FOR XTS TO RECEIVE AN ECHO OF THE MESSAGE, XTR MUST BE RUNNING.
; IT IS INITIATED IN THE FOLLOWING MANNER:
;
;       >XTR
;       LINE: DUP-0
;
; WHEN FINISHED WITH XTS/XTR, XTR MUST BE ABORTED
;-


.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
```

(continued on next page)

```
        .SBTTL  CONSTANTS
;
; RECEIVE BUFFER SIZE
;
        BUFSIZ = 90.
;
; LUN ASSIGNMENTS:
;
        TILUN=1                         ;LUN FOR TI
        CHNLUN=2                        ;LUN FOR ERROR FREE CHANNEL
        ERRLUN=3                        ;LUN FOR ERRORS
;
; EVENT FLAG ASSIGNMENTS:
;
        TIEFN=1                         ;EVENT FLAG FOR TERMINAL I/O
        CHNEFN=2                        ;EVENT FLAG FOR CHANNEL
        ERREFN=3                        ;EVENT FLAG FOR ERROR MESSAGES
        DONE=4                          ;EVENT FLAG SIGNALING COMPLETION


        .SBTTL  DATA

;
; DEFINE GCL PARAMETERS
;
        GCLDF$  TILUN,TIEFN,<LINE>,R1BUF,BUFSIZ

;****
; DPB'S
;****

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

;
; CHANNEL I/O STATUS BLOCK
;
CHNSB:  .BLKW   2
;
; TEMP LOCATION TO CONTAIN IOSB ADDRESS
;
IOSB:   .BLKW   1
TEMP:   .BLKW   1

;
; CHANNEL 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
```

(continued on next page)

```
;
; CHANNEL 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
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 MESSAGES RECEIVED OVER THE SAME LINE
;
; INPUTS:
;       NONE.
;
; OUTPUTS:
;       LOOP ALL MESSAGES INDEFINITELY.
;-

XTREP::
        CLR     R3
;
; ASSIGN LUN TO CHANNEL
;
        ALUN$S  #CHNLUN,#"NX,#0
        BCC     10$                     ;IF CC, ALL OKAY
        EPRINT  DLXERR                  ;ELSE, ASSUME DLX NOT LOADED
        BR      99$                     ;AND LEAVE
```

(continued on next page)

```
;
; 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. MAKE BELIEVE 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 IS CALLED TO GET A COMMAND LINE FOR XTR. INPUT CAN BE
; FROM TI: OR AN INDIRECT COMMAND FILE. RETURN WITH C-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.
;-
```

(continued on next page)

```
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

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$:    SEC                             ;SET-C
        BR      50$                     ;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
        .SBTTL  HNGRC2 - HANG SECOND ASYNCHRONOUS READ
;+
; **-HNGREC - HANG AN ASYNCHRONOUS READ ON THE CHANNEL
; **-HNGRC2 - HANG SECOND ASYNCHRONOUS READ ON CHANNEL
;
; 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
```

(continued on next page)

```
        .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 CHANNEL READ COMPLETE
;+
; **-RECAST - AST FOR CHANNEL READ COMPLETE
;
; INPUTS:
;       (SP) = ADDRESS OF I/O STATUS BLOCK
;
; OUTPUTS:
;       1. ANOTHER READ HUNG ON CHANNEL
;       2. BUFFER READ FROM CHANNEL IS ECHOED OVER LINE
;
;-

RECAST:
        MOV     (SP),TEMP               ; SAVE IOSB ADDRESS
        MOV     R1,(SP)                 ; SAVE R1
        MOV     TEMP,R1                 ; R1 -> IOSB
        TSTB    (R1)                    ; SUCESSFUL COMPLETION ?
        BPL     10$                     ; IF PL, YES - XMIT THE MESSAGE
        TST     R3                      ; BEEN THRU 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 IOT
5$:     TSTB    CHNSB                   ; SUCCESS ?
        BPL     20$                     ; YES - CONTINUE
        IOT                             ;  ELSE FATAL ERROR - IOT
10$:    CLR     R3                      ; CLEAR FLAG
        MOV     2(R1),XMT+Q.IOPL+2      ; SET LENGTH OF BUFFER TO XMIT
        BEQ     20$                     ; IF EQ, NO BUFFER TO XMIT ???
        MOV     4(R1),XMT+Q.IOPL        ; SET ADDRESS OF BUFFER
        CALL    XMIT                    ; ECHO MESSAGE BACK OVER LINE
20$:    CALL    @6(R1)                  ; HANG ANOTHER RECEIVE ON CHANNEL
                                        ; IGNORE ANY ERRORS
        MOV     (SP)+,R1                ; RESTORE R1
        ASTX$S                          ; EXIT AST
```

(continued on next page)

```
        .SBTTL  $EPRINT -- PRINT ERROR MESSAGE
;+
; **-$EPRINT-PRINT ERROR MESSAGE
;
; PRINTS THE SPECIFIED ERROR MESSAGE PREFIXED BY "XTR -- ".
; 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$:     MOV     R0,ERDPB+Q.IOPL         ;SET ADDRESS OF MESSAGE
10$:    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
```

4.3.10 DLX QIO Programming Example (for Ethernet device)

The XTR-DLX RECEIVER program uses DLX QIOs to receive data.

NOTE

This programming example is also included in your tape or disk kit.

```
        .TITLE  DLXRCV  - DLX Receive Program for Ethernet
        .IDENT  /V01.01/
        .ENABL  LC
        .NLIST  BEX

;
; Copyright (C) 1982, 1985 by
; DIGITAL EQUIPMENT CORPORATION, Maynard, MASS.
;
; This software is furnished under a license for use only on a
; single computer system and  may  be  copied  only  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  except for use  on  such
; system and to one who agrees to these license  terms.  Title
; to and ownership of the software shall at all  times  remain
; in DEC.
;
; The information in this document is subject to change without
; notice and should not be construed as a commitment by Digital
; Equipment Corporation.
;
; DEC assumes no responsibility for the use or  reliability of
; its software on equipment which is not supplied by DEC.
;

;+
; Module Description:
;       DLXRCV - DLX receive program for Ethernet
;       The following devices are currently supported:
;       UNA
;       QNA
;
; This program is used to monitor messages sent by other stations
; on the ethernet. Due to the data rates that are present, it can
; only be used to watch a single protocol type at a time and will
; not be able to keep up if the output device is a terminal or
; printer. This program can not be used concurrently with DECnet
; (XPT) running.
;
;    To assemble this program use the following command line:
;  MAC DLXRCV,DLXRCV/-SP=IN:[130,10]NETLIB/ML,IN:[200,200]DLXRCV
;
;    Note: The IN: device must be the DECnet distribution device
;          after the PREGEN (if any) has been performed.
;
;    To build the program:
;  TKB DLXRCV/PR:0,DLXRCV/-SP=DLXRCV
;
; Ident History:
; 1.00  21-Jul-83
;       Version 1.0 Release
;
; 1.01  11-Mar-85
;       Correct build UICs
;-
```

(continued on next page)

```
        .MCALL  EXIT$S, QIOW$S, QIO$S,  ALUN$S
        .MCALL  ASTX$S, SETF$S, CLEF$S, WTSE$S
        .MCALL  DSAR$S, ENAR$S, SREA$S, QIO$

        .MCALL  GCMLB$, NMBLK$, GCML$,  OFNB$,  OFNB$W
        .MCALL  CSI$,   CSI$SW, CSI$ND, CSI$1,  CSI$2
        .MCALL  FDAT$R, FDRC$R, FDBF$R, FCSMC$, PUT$S

        .MCALL  DLXDF$, EPMDF$, CHRDF$

        FCSMC$
        DLXDF$                          ; Define the DLX symbols
        CHRDF$                          ; Define Get/Set Char symbols
        EPMDF$                          ;
;
; Local program constants
;
        NETLUN  =       1               ; Network LUN
        OUTLUN  =       2               ; Data output LUN
        OUTL    =       144.            ; Length of output line
        TILUN   =       5               ; Terminal I/O LUN
        TILEN   =       144.            ; Terminal I/O Buffer size
        RBUFL   =       1518.           ; Receive buffer byte count
        RBCNT   =       20.             ; Number of receive buffers
        REST    =       18.             ; Rest of the receive line


        .SBTTL  Define local macros
        TAGNUM=0

        .MACRO  TYPE    MSG
        TAGNUM=TAGNUM+1
        .NLIST
        .IRP    N,<\TAGNUM>
        .SAVE
        .PSECT  DATA1,D
BUF'N:  .ASCII  "MSG"
BUFL'N  =       .-BUF'N
        .EVEN
        .RESTORE
        MOV     R0,-(SP)        ; Save the registers
        MOV     R1,-(SP)        ; ...
        MOV     R2,-(SP)        ; ...
        MOV     #BUF'N,R1       ; Set up pointers for message
        MOV     #BUFL'N,R0      ; ...
        MOV     #40,R2          ; Forms control character
        JSR     PC,PRINT        ; Go print it
        MOV     (SP)+,R2        ; Restore registers
        MOV     (SP)+,R1        ; ...
        MOV     (SP)+,R0        ; ...
        .ENDM
        .LIST
        .ENDM   TYPE
```

(continued on next page)

```
        .MACRO  ERROR   MSG
        TAGNUM=TAGNUM+1
        .NLIST
        .IRP    N,<\TAGNUM>
        .SAVE
        .PSECT  DATA1,D
BUF'N:  .ASCII  "MSG"
BUFL'N  =       .-BUF'N
        .EVEN
        .RESTORE
        MOV     R0,-(SP)        ; Save the registers
        MOV     R1,-(SP)        ; ...
        MOV     R2,-(SP)        ; ...
        MOV     #BUF'N,R1       ; Set up pointers for message
        MOV     #BUFL'N,R0      ; ...
        MOV     #60,R2          ; Forms control value
        JSR     PC,ERRPRT       ; Go print it
        MOV     (SP)+,R2        ; Restore registers
        MOV     (SP)+,R1        ; ...
        MOV     (SP)+,R0        ; ...
        .ENDM
        .LIST
        .ENDM   ERROR

        .MACRO  PROMPT  MSG
        TAGNUM=TAGNUM+1
        .NLIST
        .IRP    N,<\TAGNUM>
        .SAVE
        .PSECT  DATA1,D
BUF'N:  .ASCII  "MSG"
BUFL'N  =       .-BUF'N
        .EVEN
        .RESTORE
        MOV     #BUF'N,R1       ; Set up pointers for message
        MOV     #BUFL'N,R0      ; ...
        JSR     PC,ASK          ; Go do it
        .ENDM
        .LIST
        .ENDM   PROMPT
        .MACRO  PRO     TYP
        .LIST
        .WORD   CC.DST                  ; Characteristics type
        .WORD   4                       ; Input data byte count
        .WORD   0                       ; Output data byte count
        .WORD   0                       ; Output status
        .WORD   TYP                     ; Protocol type
        .WORD   LF$EXC!LF$DEF           ; Protocol Flags
        .NLIST
        .ENDM   PRO

        .MACRO  MUL     AD1,AD2,AD3
        .LIST
        .WORD   CC.MCT                  ; Enable multicast address for this prot
        .WORD   6                       ; Input data byte count
        .WORD   0                       ; Output data byte count
        .WORD   0                       ; Output status
        .WORD   AD1                     ; Multicast address
        .WORD   AD2                     ; ...
        .WORD   AD3                     ; ...
        .NLIST
        .ENDM   MUL

        .SBTTL  Data Area
        .PSECT  DATA2,D
```

(continued on next page)

```
;
; For information purposes - Defined in CHRDF$
;
; Protocol types
;
;CP.LOO  =       000220          ; Loopback (cross-company)      90-00
;CP.DUM  =       000540          ; Dump/Load (Digital)           60-01
;CP.CON  =       001140          ; Remote console (Digital)      60-02
;CP.ROU  =       001540          ; Routing (Digital)             60-03

;
; Multicast addresses
;
;CM.LO1  =       000317          ; First word of Loopback        CF-00
;CM.LO2  =       000000          ; Second word                   00-00
;CM.LO3  =       000000          ; Third word                    00-00

;CM.DM1  =       000253          ; First word of Dump/Load       AB-00
;CM.DM2  =       000400          ; Second word                   00-01
;CM.DM3  =       000000          ; Third word                    00-00

;CM.RC1  =       000253          ; First word of Remote console  AB-00
;CM.RC2  =       001000          ; Seconds word                  00-02
;CM.RC3  =       000000          ; Third word                    00-00
;
; Characteristics buffers
;
.LIST   ME
CHRLOO:
        PRO     CP.LOO                  ; Loopback
        MUL     CM.LO1,CM.LO2,CM.LO3    ; Loopback Multicast
LOOLEN = .-CHRLOO

CHRDLL:
        PRO     CP.DUM                  ; Dump/Load
        MUL     CM.DM1,CM.DM2,CM.DM3
DLLLEN = .-CHRDLL

CHRRCN:
        PRO     CP.CON                  ; Remote console
        MUL     CM.RC1,CM.RC2,CM.RC3
RCNLEN = .-CHRRCN

CHRROU:
        PRO     CP.ROU                  ; Router
        MUL     253,1400,0
        MUL     253,2000,0
        MUL     243,1400,0
        MUL     243,2000,0
ROULEN = .-CHRROU


DEVMDE: .WORD   CC.ECM                  ; Ethernet channel mode
        .WORD   2                       ; Two bytes of data
        .WORD   0                       ; Data bytes out
        .WORD   0                       ; Status
        .WORD   0                       ; Mode word for device
MDELEN = .-DEVMDE

PRMMDE = 100000                         ; Promiscuous mode for UNA
```

(continued on next page)

```
        .SBTTL  Receive buffers
        .PSECT  DATA3,D
        .NLIST  ME

        RB=0
        .REPT   RBCNT                   ; Create the receive buffers
        RB=RB+1
        .IRP    N,<\RB>
;
; Receive buffer #'N
;
RBUF'N: .BLKB   RBUFL                   ; Receive buffer
;
; Buffer for DLX receive set characteristics
;
CHR'N:
        .WORD   CC.DAD                  ; Read destination address
        .WORD   6                       ; Input data byte count
        .WORD   0                       ; Output data byte count
        .WORD   0                       ; Status
        .WORD   0                       ; Ethernet address
        .WORD   0                       ; ...
        .WORD   0                       ; ...
        .WORD   CC.ADR                  ; Read source Address
        .WORD   6                       ; Input data byte count
        .WORD   0                       ; Output data byte count
        .WORD   0                       ; Status
        .WORD   0                       ; Ethernet address
        .WORD   0                       ; ...
        .WORD   0                       ; ...
        .WORD   CC.PRO                  ; Read protocol type
        .WORD   2                       ; Input data byte count
        .WORD   0                       ; Output data byte count
        .WORD   0                       ; Status
        .WORD   0                       ; Protocol type
CHRL'N  = .-CHR'N

        .ENDM
        .ENDR

        IO=0
        .REPT   RBCNT
        IO=IO+1
        .IRP    N,<\IO>

BSB'N:.BLKW     2                       ; Rec buffer  IOSB
        .WORD   0                       ; Link word to next descriptor
        .WORD   RBUF'N                  ; Address of the receive buffer
        .WORD   CHR'N                   ; Address of the characteristics buffer
BSBL'N = .-BSB'N

        .ENDM
        .ENDR
         .PSECT  DATA4,D                ; Local storage area

 NXIOSB: .BLKW   2                      ; Network I/O status doubleword

 TIOSB:  .BLKW   2                      ; Terminal I/O Status Doubleword

 OUTSB:  .BLKW   2                      ; Output IOSB

 BUFCNT: .WORD   0                      ; Local count of received packets
```

(continued on next page)

```
RCVQUE: .WORD   0                       ; Receive queue listhead
        .WORD   RCVQUE                  ; ...
        .WORD   -1                      ; Temp flag

TIBUF:  .BLKB   TILEN                   ; Terminal I/O Buffer

OUTBUF:
        .BLKB   OUTL                    ; Output buffer

DEF:    .ASCII  /UNA-0/
DEFL    =       .-DEF

CNTSTR: .ASCII  / Total number of packets received /
        CNTL = .-CNTSTR

        .EVEN



        .SBTTL  CONSTANTS FOR PROGRAM AND FILE BLOCK ALLOCATION
;
        MS1.TB=1
;

        FSRSZ$  2                       ; File buffers
FDBOUT:
        FDBDF$                          ; Allocate space for FDB
        FDAT$A  R.VAR,FD.CR             ; File attributes
        FDRC$A  ,OUTBUF,OUTL            ; Record access mode
        FDOP$A  OUTLUN,DFOUT            ; File open section
;
;
LCSIO:
        CSI$
        .BLKB   C.SIZE                  ; Allocates required storage.
        .EVEN
DFOUT:
        NMBLK$  ETHER,DAT,,SY,0


SWTABU:
        CSI$SW  TB,MS1.TB               ; Tab switch
        CSI$ND                          ; End of table

        .EVEN
        .SBTTL  Main Code
        .PSECT  CODE
        .ENABL  LSB
;
; QIO directives
;
XIORCV: QIO$    IO.XRC,NETLUN,,,0,RCVAST,<0,RBUFL,0,CHRL1>
OUTIO:  QIO$    IO.WVB,OUTLUN,2,,OUTSB,,<OUTBUF,OUTL,60>

START:
        FINIT$                          ; Initialize files
        ALUN$S  #TILUN,#"TI             ; Assign LUN to terminal
        TYPE    < UNA Ethernet monitor >
        TYPE    <  >
        TYPE    < The default output is SY:ETHER.DAT >
```

(continued on next page)

```
        TYPE    < >
        PROMPT  <Output Device? (File, TI: or Return) >
        MOV     R1,LCSIO+C.CMLD+2
        MOV     R0,LCSIO+C.CMLD
        CSI$1   #LCSIO
        CSI$2   #LCSIO,OUTPUT,#SWTAB0
        FDOP$R  #FDBOUT,#OUTLUN,#LCSIO+C.DSDS,#DFOUT
        OPEN$W  #FDBOUT
        BCC     10$
        ERROR   < Could not open file >
        JMP     ERREXT
10$:
        ALUN$S  #NETLUN,#"NX,#0         ; Assign a LUN to DLX
        BCC     20$                     ; Branch if no error
        ERROR   <Could not assign LUN to NX >
        JMP     ERREXT

20$:
        PROMPT  < Enter line (UNA-0): >
        BNE     30$                     ; Branch if response
        MOV     #DEF,R1                 ; Get default answer
        MOV     #DEFL,R0                ; And default length

;
; Open the Line
;
30$:
        MOV     #NXIOSB,R3              ; Point to network IOSB
        QIOW$S  #IO.XOP,#NETLUN,#1,,R3,,<R1,R0,#400>
        TSTB    (R3)                    ; Get "open" QIO status
        BPL     40$                     ; Branch if no error
        ERROR   < Could not open line >
        CALL    PRTOB                   ; Print the error code
        JMP     ERREXT                  ; Get out
;
; Get current mode
;
40$:
        MOV     #DEVMDE,R4              ; Get address of request buffer
        QIOW$S  #IO.XGC,#NETLUN,#1,,R3,,<R4,#MDELEN>
        TSTB    (R3)                    ; Was request successful?
        BPL     50$                     ; If PL, Yes
        ERROR   < QIO for get device mode failed>
        JMP     ERREXT
50$:
        CMP     #CS.SUC,6(R4)           ; Was this request successfull?
        BEQ     60$                     ; If EQ,yes
        ERROR   < Get device mode request failed>
        JMP     ERREXT
;
; Set device in promiscuous mode
;
60$:
        BIS     #PRMMDE,10(R4)          ; Set promiscuous mode
        QIOW$S  #IO.XSC,#NETLUN,#1,,R3,,<R4,#MDELEN>
        TSTB    (R3)                    ; Was request successful
        BPL     70$                     ; If PL, Yes
        ERROR   < QIO for set device mode failed>
        JMP     ERREXT
70$:
        CMP     #CS.SUC,6(R4)           ; Was this request successfull?
        BEQ     80$                     ; If EQ,yes
        ERROR   < Set device mode request failed>
        JMP     ERREXT
```

(continued on next page)

```
80$:
        PROMPT  <What do you want to monitor (Loop,DLL/DUM,Console,Router)>
        MOV     #CHRLOO,R4              ; Assume loopback
        MOV     #LOOLEN,R5              ; ...
        CMPB    #'L,(R1)                ; Do they want loop
        BEQ     90$                     ; If EQ, Yes
        MOV     #CHRDLL,R4              ; Try DLL/DUM
        MOV     #DLLLEN,R5              ; ...
        CMPB    #'D,(R1)                ; Do they want DLL?
        BEQ     90$                     ; If EQ,yes
        MOV     #CHRRCN,R4              ; Remote console
        MOV     #RCNLEN,R5              ; ...
        CMPB    #'C,(R1)                ; If EQ, remote console will be displaye
        BEQ     90$
        MOV     #CHRROU,R4              ; How about router
        MOV     #ROULEN,R5              ; ...
        CMPB    #'R,(R1)
        BNE     80$                     ; If NE, ask again
        TYPE    <Warning!! I may not be able to keep up on loaded networks!>

;
; Enable protocol type to monitor
;
90$:
        MOV     #NXIOSB,R3              ; Get IOSB address
        QIOW$S  #IO.XSC,#NETLUN,#1,,R3,,<R4,R5> ; Set the protocol type
        TSTB    (R3)                    ; Get "SET CHR" QIO status
        BMI     120$                    ; Branch if error
100$:
        CMP     #CS.SUC,6(R4)           ; Was this request successfull?
        BNE     110$                    ; If NE, no
        SUB     2(R4),R5                ; Length of request
        SUB     #10,R5                  ; Length of header
        BLE     140$                    ; All done
        ADD     2(R4),R4                ; Point to next request
        ADD     #10,R4                  ; ...
        BR      100$                    ; Check next one
110$:
        MOV     R4,-(SP)                ; Save request address
        ERROR   < Set characteristics error - Bad status in buffer >
        MOV     (SP)+,R4                ; Restore the request address
        MOVB    6(R4),R3                ; Get the error code
        CALL    PRTOB                   ; Print the low byte
        MOVB    7(R4),R3                ; Get the high byte
        BR      130$
120$:
        ERROR   < Set characteristics error - Set protocol (QIO status) >
130$:
        CALL    PRTOB                   ; Print the error code
        JMP     EXIT                    ; Get out
;
; Post the receive buffers
;
140$:
        MOV     #BSB1,R1                ; Pointer to first IOSB
        MOV     #RBCNT,R0               ; Number of receive buffers
        DSAR$S                          ; Disable AST's
```

(continued on next page)

```
150$:
        MOV     R1,4(R1)                ; Fill in link word address
        ADD     #BSBL1,4(R1)            ; of next descriptor
        MOV     R1,Q.IOSB+XIORCV        ; Fill in IOSB field
        MOV     6(R1),Q.IOPL+XIORCV     ; Fill in buffer address
        MOV     10(R1),Q.IOPL+4+XIORCV  ; Fill in characteristics buffer address
        DIR$    #XIORCV                 ; Issue the receive request
        BCS     190$                    ; If CS, error with directive
        ADD     #BSBL1,R1               ; Point to next IOSB
        DEC     R0                      ; One less to post
        BNE     150$                    ; Loop till done
        SUB     #BSBL1,R1               ; Backup pointer to last descriptor
        MOV     #BSB1,4(R1)             ; Change to point to first descriptor
        CLEF$S  #5                      ; Clear the event flag
        ENAR$S                          ; Enable AST's
        SREA$S  #ABOAST                 ; Specify an abort AST
        TYPE    <Monitor Starting...>
160$:
        MOV     4(R1),R1                ; Get next descriptor address
        TST     (R1)                    ; Has buffer been used?
        BNE     170$                    ; If NE, Yes
        WTSE$S  #5                      ; Wait for event 5 (RCV complete)
;
; Process the receive buffers
;
170$:
        CLEF$S  #5                      ; Clear the event flag
        INC     BUFCNT                  ; Another one received
        TSTB    (R1)                    ; Did rec complete successfully?
        BMI     200$                    ; If MI, Repost the buffer
        MOV     #OUTBUF,R0              ; Output buffer area
        MOV     10(R1),R3               ; Get address of the char buffer
        ADD     #C.CHRL,R3              ; Point to destination address
        MOV     #6,R4                   ; Number of bytes to convert
        CALL    CNV                     ; Convert to ASCII
        MOVB    #40,(R0)+               ; Add another space
        ADD     #C.CHRL,R3              ; Point to source address
        MOV     #6,R4                   ; Number of bytes to convert
        CALL    CNV                     ; Convert to ASCII
        MOVB    #40,(R0)+               ; Add another space
        ADD     #C.CHRL,R3              ; Point to protocol type
        MOV     #2,R4                   ; Number of bytes to convert
        CALL    CNV                     ; Convert to ASCII
        MOVB    #40,(R0)+               ; A couple of more spaces
        MOVB    #40,(R0)+               ; ...

        MOV     #REST,R4                ; Print out the rest of the line
        MOV     6(R1),R3                ; Get address of the data buffer
        CALL    CNV                     ; Convert the data to ascii
180$:
        CALL    POST                    ; Post another receive

        PUT$S   #FDBOUT
        BCC     160$                    ; Check for error
        ERROR   < Device write error >
        BR      EXIT                    ; Print error and get out

190$:
        ERROR   < Receive directive error >
        MOV     R1,R3                   ; Get IOSB
        CALL    PRTOB                   ; Print the error code
        BR      EXIT                    ; Get out
```

(continued on next page)

```
200$:
        ERROR   < Receive buffer error >
        MOV     R1,R3                   ; Point to IOSB
        CALL    PRTOB                   ; Print the error code
        CALL    POST                    ; Post the receive again
        BR      160$                    ; Wait for another completion
        .DSABL  LSB


EXIT:
        CLOSE$  #FDBOUT
        MOV     #DEVMDE,R4              ; Get address of request buffer
        BIC     #PRMMDE,10(R4)          ; Clear promiscuous mode
        QIOW$S  #IO.XSC,#NETLUN,#1,,R3,,<R4,#MDELEN> ; ...
        QIOW$S  #IO.XCL,#NETLUN,#1      ; Close the opened line
ERREXT:
        CLOSE$  #FDBOUT
        EXIT$S                          ; Program exits

POST:
        CLR     (R1)                    ; Clear out the IOSB
        CLR     2(R1)                   ; ...
        MOV     R1,Q.IOSB+XIORCV        ; Fill in IOSB address
        MOV     6(R1),Q.IOPL+XIORCV     ; Fill in buffer address
        MOV     10(R1),Q.IOPL+4+XIORCV  ; Fill in characteristics buffer address
        DIR$    #XIORCV                 ; Post the buffer
        RETURN
        .SBTTL  AST routines
;+
; RCVAST Receive buffer AST routine
;-
RCVAST:
        TST     (SP)+                   ;; Remove IOSB address
        SETF$S  #5                      ;; Set event flag for rcv complete
        ASTX$S                          ;; Exit from AST


;+
; ABOAST - AST routine for abort requests
;-
ABOAST:
        MOV     #OUTBUF,R0              ; Get output buffer pointer
        MOV     #CNTSTR,R1              ; Address of count text
        MOV     #CNTL,R2                ; Get the length
10$:
        MOVB    (R1)+,(R0)+             ; Move the characters
        DEC     R2
        BNE     10$

        MOV     BUFCNT,R1               ; Get the packet count
        CLR     R2                      ; Suppress leading zeros
        CALL    $CBDMG                  ; Convert to decimal count
        MOV     #OUTL,R1                ; Compute the space left
        ADD     #OUTBUF,R1
        SUB     R0,R1
20$:
        MOVB    #40,(R0)+               ; Fill rest with blanks
        DEC     R1
        BNE     20$

        PUT$S   #FDBOUT
        QIOW$S  #IO.WVB,#TILUN,#7,,,,<#OUTBUF,#OUTL,#60>
        JMP     EXIT                    ; Go close the line
```

(continued on next page)

```
        .SBTTL  TERMINAL I/O SUBROUTINES
;+
; ASK - Prompt and get response
;
; Input:
;       R0 - Length of prompt
;       R1 - Address of prompt string
;
; Output:
;       R0 - Length of response
;       R1 - Address of response string
;
;       Z bit set if nothing typed (just a carrige return)
;-
ASK:
        QIOW$S #IO.RPR,#TILUN,#9.,,#TIOSB,,<#TIBUF,#TILEN,,R1,R0,#44>
        CMPB    TIOSB,#IE.EOF           ; Was a Ctrl/Z typed?
        BNE     10$                     ; Branch if no
        SEC                             ; Indicate ^Z typed
        RETURN
10$:
        MOV     #TIBUF,R1               ; Get buffer address
        MOV     TIOSB+2,R0              ; Save byte count in R0
        CLC                             ; Clear the carry
        RETURN

;+
; PRTOB  - Print octal data byte
;
; Inputs:
;       R3 - Data Buffer Address
;
; Outputs:
;       Byte printed on TI
;       R0,R2 - Destroyed
;-
PRTOB:
        MOV     #TIBUF,R0               ; Output area
        MOVB    (R3)+,R2                ; Get data byte
        BIC     #^C<377>,R2             ; Clear out high byte
        ASH     #2,R2                   ; Convert to table offset
        ADD     #CNVTAB,R2              ; Add in base address of table
        MOVB    (R2)+,(R0)+             ; Get the conversion data
        MOVB    (R2)+,(R0)+             ; ...
        MOVB    (R2)+,(R0)+             ; ...

        QIOW$S  #IO.WVB,#TILUN,#9.,,,,<#TIBUF,#3,#40> ; Print the data
        RETURN
;+
; ERRPRT - Print error messages
; PRINT  - Print a message on the terminal
;
; Inputs:
;       R0 - Length of string
;       R1 - Address of error message
;       R2 - Forms control value
;-
ERRPRT:
PRINT:
        QIOW$S  #IO.WVB,#TILUN,#9.,,,,<R1,R0,R2>
        RETURN
```

(continued on next page)

```
;+
; CNV - Convert binary to ASCII string
;
; Inputs:
;       R0 -> Output buffer area
;       R3 -> Input buffer of binary data
;       R4 - Number of bytes to convert
; Outputs:
;       R0 -> Next available address in the output area
;       R3 -> Next byte after "count" bytes are converted
;       R4 - 0
; Destroyed R2
;-
CNV:
        MOVB    (R3)+,R2                ; Get the data byte
        BIC     #^C<377>,R2             ; Clear out high byte
        ASH     #2,R2                   ; Shift left twice
        ADD     #CNVTAB,R2              ; Add start of table
        MOVB    (R2)+,(R0)+             ; Move the first character
        MOVB    (R2)+,(R0)+             ; Second
        MOVB    (R2)+,(R0)+             ; Third
        MOVB    (R2)+,(R0)+             ; Fourth
        SOB     R4,CNV                  ; Loop till done
        RETURN

CNVTAB:
        .BYTE   60,60,60,40     ;000
        .BYTE   60,60,61,40     ;001
        .BYTE   60,60,62,40     ;002
        .BYTE   60,60,63,40     ;003
        .BYTE   60,60,64,40     ;004
        .BYTE   60,60,65,40     ;005
.NLIST                                  ; ETC...
        .BYTE   60,60,66,40     ;006
        .BYTE   60,60,67,40     ;007
        .BYTE   60,61,60,40     ;010
        .BYTE   60,61,61,40     ;011
        .BYTE   60,61,62,40     ;012
        .BYTE   60,61,63,40     ;013
        .BYTE   60,61,64,40     ;014
        .BYTE   60,61,65,40     ;015
        .BYTE   60,61,66,40     ;016
        .BYTE   60,61,67,40     ;017
        .BYTE   60,62,60,40     ;020
        .BYTE   60,62,61,40     ;021
        .BYTE   60,62,62,40     ;022
        .BYTE   60,62,63,40     ;023
        .BYTE   60,62,64,40     ;024
        .BYTE   60,62,65,40     ;025
        .BYTE   60,62,66,40     ;026
        .BYTE   60,62,67,40     ;027
        .BYTE   60,63,60,40     ;030
        .BYTE   60,63,61,40     ;031
        .BYTE   60,63,62,40     ;032
        .BYTE   60,63,63,40     ;033
        .BYTE   60,63,64,40     ;034
        .BYTE   60,63,65,40     ;035
        .BYTE   60,63,66,40     ;036
        .BYTE   60,63,67,40     ;037
        .BYTE   60,64,60,40     ;040
        .BYTE   60,64,61,40     ;041
        .BYTE   60,64,62,40     ;042
```

(continued on next page)

```
.BYTE   60,64,63,40     ;043
.BYTE   60,64,64,40     ;044
.BYTE   60,64,65,40     ;045
.BYTE   60,64,66,40     ;046
.BYTE   60,64,67,40     ;047
.BYTE   60,65,60,40     ;050
.BYTE   60,65,61,40     ;051
.BYTE   60,65,62,40     ;052
.BYTE   60,65,63,40     ;053
.BYTE   60,65,64,40     ;054
.BYTE   60,65,65,40     ;055
.BYTE   60,65,66,40     ;056
.BYTE   60,65,67,40     ;057
.BYTE   60,66,60,40     ;060
.BYTE   60,66,61,40     ;061
.BYTE   60,66,62,40     ;062
.BYTE   60,66,63,40     ;063
.BYTE   60,66,64,40     ;064
.BYTE   60,66,65,40     ;065
.BYTE   60,66,66,40     ;066
.BYTE   60,66,67,40     ;067
.BYTE   60,67,60,40     ;070
.BYTE   60,67,61,40     ;071
.BYTE   60,67,62,40     ;072
.BYTE   60,67,63,40     ;073
.BYTE   60,67,64,40     ;074
.BYTE   60,67,65,40     ;075
.BYTE   60,67,66,40     ;076
.BYTE   60,67,67,40     ;077
.BYTE   61,60,60,40     ;100
.BYTE   61,60,61,40     ;101
.BYTE   61,60,62,40     ;102
.BYTE   61,60,63,40     ;103
.BYTE   61,60,64,40     ;104
.BYTE   61,60,65,40     ;105
.BYTE   61,60,66,40     ;106
.BYTE   61,60,67,40     ;107
.BYTE   61,61,60,40     ;110
.BYTE   61,61,61,40     ;111
.BYTE   61,61,62,40     ;112
.BYTE   61,61,63,40     ;113
.BYTE   61,61,64,40     ;114
.BYTE   61,61,65,40     ;115
.BYTE   61,61,66,40     ;116
.BYTE   61,61,67,40     ;117
.BYTE   61,62,60,40     ;120
.BYTE   61,62,61,40     ;121
.BYTE   61,62,62,40     ;122
.BYTE   61,62,63,40     ;123
.BYTE   61,62,64,40     ;124
.BYTE   61,62,65,40     ;125
.BYTE   61,62,66,40     ;126
.BYTE   61,62,67,40     ;127
.BYTE   61,63,60,40     ;130
.BYTE   61,63,61,40     ;131
.BYTE   61,63,62,40     ;132
.BYTE   61,63,63,40     ;133
.BYTE   61,63,64,40     ;134
.BYTE   61,63,65,40     ;135
.BYTE   61,63,66,40     ;136
.BYTE   61,63,67,40     ;137
```

(continued on next page)

```
.BYTE   61,64,60,40     ;140
.BYTE   61,64,61,40     ;141
.BYTE   61,64,62,40     ;142
.BYTE   61,64,63,40     ;143
.BYTE   61,64,64,40     ;144
.BYTE   61,64,65,40     ;145
.BYTE   61,64,66,40     ;146
.BYTE   61,64,67,40     ;147
.BYTE   61,65,60,40     ;150
.BYTE   61,65,61,40     ;151
.BYTE   61,65,62,40     ;152
.BYTE   61,65,63,40     ;153
.BYTE   61,65,64,40     ;154
.BYTE   61,65,65,40     ;155
.BYTE   61,65,66,40     ;156
.BYTE   61,65,67,40     ;157
.BYTE   61,66,60,40     ;160
.BYTE   61,66,61,40     ;161
.BYTE   61,66,62,40     ;162
.BYTE   61,66,63,40     ;163
.BYTE   61,66,64,40     ;164
.BYTE   61,66,65,40     ;165
.BYTE   61,66,66,40     ;166
.BYTE   61,66,67,40     ;167
.BYTE   61,67,60,40     ;170
.BYTE   61,67,61,40     ;171
.BYTE   61,67,62,40     ;172
.BYTE   61,67,63,40     ;173
.BYTE   61,67,64,40     ;174
.BYTE   61,67,65,40     ;175
.BYTE   61,67,66,40     ;176
.BYTE   61,67,67,40     ;177
.BYTE   62,60,60,40     ;200
.BYTE   62,60,61,40     ;201
.BYTE   62,60,62,40     ;202
.BYTE   62,60,63,40     ;203
.BYTE   62,60,64,40     ;204
.BYTE   62,60,65,40     ;205
.BYTE   62,60,66,40     ;206
.BYTE   62,60,67,40     ;207
.BYTE   62,61,60,40     ;210
.BYTE   62,61,61,40     ;211
.BYTE   62,61,62,40     ;212
.BYTE   62,61,63,40     ;213
.BYTE   62,61,64,40     ;214
.BYTE   62,61,65,40     ;215
.BYTE   62,61,66,40     ;216
.BYTE   62,61,67,40     ;217
.BYTE   62,62,60,40     ;220
.BYTE   62,62,61,40     ;221
.BYTE   62,62,62,40     ;222
.BYTE   62,62,63,40     ;223
.BYTE   62,62,64,40     ;224
.BYTE   62,62,65,40     ;225
.BYTE   62,62,66,40     ;226
.BYTE   62,62,67,40     ;227
.BYTE   62,63,60,40     ;230
.BYTE   62,63,61,40     ;231
.BYTE   62,63,62,40     ;232
.BYTE   62,63,63,40     ;233
.BYTE   62,63,64,40     ;234
```

(continued on next page)

```
.BYTE   62,63,65,40     ;235
.BYTE   62,63,66,40     ;236
.BYTE   62,63,67,40     ;237
.BYTE   62,64,60,40     ;240
.BYTE   62,64,61,40     ;241
.BYTE   62,64,62,40     ;242
.BYTE   62,64,63,40     ;243
.BYTE   62,64,64,40     ;244
.BYTE   62,64,65,40     ;245
.BYTE   62,64,66,40     ;246
.BYTE   62,64,67,40     ;247
.BYTE   62,65,60,40     ;250
.BYTE   62,65,61,40     ;251
.BYTE   62,65,62,40     ;252
.BYTE   62,65,63,40     ;253
.BYTE   62,65,64,40     ;254
.BYTE   62,65,65,40     ;255
.BYTE   62,65,66,40     ;256
.BYTE   62,65,67,40     ;257
.BYTE   62,66,60,40     ;260
.BYTE   62,66,61,40     ;261
.BYTE   62,66,62,40     ;262
.BYTE   62,66,63,40     ;263
.BYTE   62,66,64,40     ;264
.BYTE   62,66,65,40     ;265
.BYTE   62,66,66,40     ;266
.BYTE   62,66,67,40     ;267
.BYTE   62,67,60,40     ;270
.BYTE   62,67,61,40     ;271
.BYTE   62,67,62,40     ;272
.BYTE   62,67,63,40     ;273
.BYTE   62,67,64,40     ;274
.BYTE   62,67,65,40     ;275
.BYTE   62,67,66,40     ;276
.BYTE   62,67,67,40     ;277
.BYTE   63,60,60,40     ;300
.BYTE   63,60,61,40     ;301
.BYTE   63,60,62,40     ;302
.BYTE   63,60,63,40     ;303
.BYTE   63,60,64,40     ;304
.BYTE   63,60,65,40     ;305
.BYTE   63,60,66,40     ;306
.BYTE   63,60,67,40     ;307
.BYTE   63,61,60,40     ;310
.BYTE   63,61,61,40     ;311
.BYTE   63,61,62,40     ;312
.BYTE   63,61,63,40     ;313
.BYTE   63,61,64,40     ;314
.BYTE   63,61,65,40     ;315
.BYTE   63,61,66,40     ;316
.BYTE   63,61,67,40     ;317
.BYTE   63,62,60,40     ;320
.BYTE   63,62,61,40     ;321
.BYTE   63,62,62,40     ;322
.BYTE   63,62,63,40     ;323
.BYTE   63,62,64,40     ;324
.BYTE   63,62,65,40     ;325
.BYTE   63,62,66,40     ;326
.BYTE   63,62,67,40     ;327
.BYTE   63,63,60,40     ;330
.BYTE   63,63,61,40     ;331
```

(continued on next page)

```
.BYTE   63,63,62,40     ;332
.BYTE   63,63,63,40     ;333
.BYTE   63,63,64,40     ;334
.BYTE   63,63,65,40     ;335
.BYTE   63,63,66,40     ;336
.BYTE   63,63,67,40     ;337
.BYTE   63,64,60,40     ;340
.BYTE   63,64,61,40     ;341
.BYTE   63,64,62,40     ;342
.BYTE   63,64,63,40     ;343
.BYTE   63,64,64,40     ;344
.BYTE   63,64,65,40     ;345
.BYTE   63,64,66,40     ;346
.BYTE   63,64,67,40     ;347
.BYTE   63,65,60,40     ;350
.BYTE   63,65,61,40     ;351
.BYTE   63,65,62,40     ;352
.BYTE   63,65,63,40     ;353
.BYTE   63,65,64,40     ;354
.BYTE   63,65,65,40     ;355
.BYTE   63,65,66,40     ;356
.BYTE   63,65,67,40     ;357
.BYTE   63,66,60,40     ;360
.BYTE   63,66,61,40     ;361
.BYTE   63,66,62,40     ;362
.BYTE   63,66,63,40     ;363
.BYTE   63,66,64,40     ;364
.BYTE   63,66,65,40     ;365
.BYTE   63,66,66,40     ;366
.BYTE   63,66,67,40     ;367
.BYTE   63,67,60,40     ;370
.BYTE   63,67,61,40     ;371
.BYTE   63,67,62,40     ;372
.BYTE   63,67,63,40     ;373
.BYTE   63,67,64,40     ;374
.BYTE   63,67,65,40     ;375
.BYTE   63,67,66,40     ;376
.BYTE   63,67,67,40     ;377

.END    START
```

A
Disconnect or Reject Reason Codes

The following list contains the error reasons codes available at the logical link user interface. These codes can be returned after either of the following events occurs:

- A connect request was rejected by the network (IE.NRJ); see Section 2.3.5.
- A connected logical link was aborted by the network (NT.ABO); see Section 2.3.2.

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

(continued on next page)

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

(continued on next page)

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

The object type code values that have been defined by Digital are listed below, 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 listed object types have a descriptor format of 0, requiring definition by the object type codes given in the first two columns below.

Object Type		
Octal	**Decimal**	**Process Type**
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

(continued on next page)

Object Type Octal	Decimal	Process Type
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
Summary of 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 you 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 has been made to open a file on an active channel. Either another channel must be used or the active channel must be released via a close prior to reusing it. The second word of the I/O status block is not applicable.
177776 (–2)	CHANNEL NOT ACTIVE A file operation request has been made on an inactive channel. Either a file open has not been issued on this channel or the network link for this channel has been lost. The second word of the I/O status block is not applicable.

(continued on next page)

Table C–1 (cont.): First Word I/O Status Block Error Codes

Error Code	Description
177775 (-3)	DATA ACCESS PROTOCOL ERROR An error has been detected by the remote file system or by the remote server task. The error is then returned to the user by DAP. The second word of the I/O status block contains the file access error code. Look up this code in Table C–3.
177774 (-4)	NSP ERROR (see Table C–2) The Data Access Protocol (DAP) utilities depend on Network Services Protocol (NSP) as a vehicle for accessing remote files. This code indicates that a problem has been 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 this 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 has been found in the attributes array (*ichar*) of an open command.
177772 (-6)	DATA OVERRUN A message or block of messages was received that 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 that should be reported 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 0 channel or a channel value greater than 255 is invalid. The second word of the I/O status block is not applicable.

(continued on next page)

Table C–1 (cont.): First Word I/O Status Block Error Codes

Error Code	Description
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, the FORTRAN program must be rebuilt, increasing the size of $FSR1 in the task build. The second word of the I/O status block is not applicable.
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 (for example, 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 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 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).

(continued on next page)

Table C–2 (cont.): NSP Error Codes

Error Code	Description
-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	A transmission of an interrupt message was attempted before the last one had finished.
-12	A connect reject was issued by the user task to which the connection was attempted.
-13	A buffer is either 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 is used to return 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 is divided into 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 located 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 that has occurred. The specific reason for the error is given in the miccode field (the low-order byte of the same word that contains the maccode field). Miccode values correlating to each maccode value listed in Table C–3 are found in the table referenced in the last column of Table C–3.

Table C–3: DAP Maccode Field Values

Field Value	Error Type	Meaning	Miccode Table
0	Pending	The operation is in progress.	C–5
1	Successful	Returns information that indicates success.	C–5
2	Unsupported	This implementation of DAP does not support specified request.	C–4
3	Reserved		
4	File open	Errors that occur before a file is successfully opened.	C–5
5	Transfer error	Errors that occur after a file is opened and before it is closed.	C–5
6	Transfer warning	For operations on open files, indicates that the operation completed, but not with complete success.	C–5
7	Access termination	Errors associated with terminating access to a file.	C–5
10	Format	Error in parsing a message. Format is not correct.	C–4
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).	C–4
12	Sync	DAP message received out of synchronization.	C–6
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 value returned in this field identifies the specific reason for the error type defined in the maccode field (see Section C.2.1). Miccode field values are defined in three different tables, each table associated with certain maccode values, as outlined below:

- 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 field where the error is located is described in the third column.

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	File 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)

(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
	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)
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)

(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
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)
Data message errors		
10	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	Record number field (RECNUM)
	21	File data field (FILEDATA)

(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
Status message errors		
11	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	Macro status code field (MACCODE)
	21	Micro status code field (MICCODE)
	22	Record file address field (RFA)
	23	Record number field (RECNUM)
	24	Secondary status field (STV)
	25	Secondary status text (STX)
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)

(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
	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)
	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)

(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
	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)
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)

(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
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)
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 (PROWLD)

(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
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)	Corresponding Symbolic Status Code	Error Description
0		Unspecified error
1	ER$ABO	Operation aborted
2	ER$ACC	F11-ACP 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$AST	Invalid (that is, synchronous) operation at AST level
12	ER$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	F11-ACP could not create file (STV = system error code)

(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)	Corresponding Symbolic Status Code	Error Description
31	ER$CUR	No current record (operation not preceded by get/find)
32	ER$DAC	F11-ACP 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	F11-ACP 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	F11-ACP 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)	Corresponding 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$MRN	MRN value = negative or relative key > MRN
112	ER$MRS	MRS value = 0 for fixed length records and/or relative files

(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)	Corresponding Symbolic Status Code	Error Description
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; no 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	F11-ACP remove function failed
140	ER$RNF	Record not found
141	ER$RNL	Record not locked
142	ER$ROP	Invalid record options

(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)	Corresponding Symbolic Status Code	Error Description
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
170	CAA	Cannot access argument list
171	CCF	Cannot close file

(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)	Corresponding Symbolic Status Code	Error Description
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 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
221	RLF	Invalid related file
222	RSS	Invalid resultant string size
223	RST	Invalid resultant string address

(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)	Corresponding Symbolic Status Code	Error Description
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
252	IE.DUN	Device not attachable
253	IE.RSU	Shareable resource in use
254	IE.OVR	Illegal overlay request

(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)	Corresponding Symbolic Status Code	Error Description
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)	Corresponding 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 not readable or writeable 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

(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)	Corresponding Symbolic Status Code	Error Description
336	RSL	Invalid resultant string length
337	RVU	Error updating RRVs; some paths to data may be lost
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

(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)	Corresponding Symbolic Status Code	Error Description
364	IDR	Invalid directory rename operation
365	STR	User structure (FAB/RAB) became invalid during operation
366	FTM	Network file transfer mode precludes operation
6000		User-defined errors to 7777

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

.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.
;
;
;       +-------------------------------+ 000
;       |            N.RND              |
;       |                               | 004
;       +-------------------------------+
;       |   N.ROT           N.RFM       | 006
;       +-------------------------------+
;
;                  FORMAT 0
;
;       +-------------------------------+ 010
;       |           (UNUSED)            |
;       |                               | 030
;       +-------------------------------+
```

(continued on next page)

```
;
;
;                       FORMAT 1
;
;       +-------------------------------+
;       |           N.RDEC              |   010
;       +-------------------------------+
;       |           N.RDE               |   012
;       +-------------------------------+   030
;       |           N.RIDC              |   032
;       +-------------------------------+
;       |           N.RID               |   034
;       +-------------------------------+   052
;       |           N.RPSC              |   054
;       +-------------------------------+
;       |           N.RPS               |   056
;       +-------------------------------+   064
;       |           N.RACC              |   066
;       +-------------------------------+
;       |           N.RAC               |   070
;       +-------------------------------+   106
;
;
;
          .MACRO    CRBDF$,L,B,LST
          .iif nb   LST       ,List
          .ASECT
.=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

.=.-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
```

(continued on next page)

```
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 block

           .PSECT
           .ENDM          CRBDF$
           .MACRO         CNBDF$,L,B,LST
           .iif nb        LST       .List
           .ASECT

.=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
```

(continued on next page)

```
$$$=.
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 coun
N.CAC:'L'   .BLKB       16.     ; Accounting information
N.CDAC:'L'  .BLKW       1       ; Optional data byte count
N.CDA:'L'                       ; Optional data
                                ;
N.CBL='B'.-N.CTL                ; Length of cnb (without any data)

.=$$$
N.CDEV:'L'  .BLKW       1       ; Default device name (from accoun
N.CUNI:'L'  .BLKB       1       ; Default device unit number
            .EVEN
N.CUIC:'L'  .BLKW       1       ; Login UIC from account file
            .PSECT
N.CDDS:'L'  .BLKB       11.     ; Default directory string
            .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 a 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 pertaining to this manual. These codes are defined in the IOERR$ macro in RSXMAC.SML, which is referenced in the NSSYM$ macro in NETLIB.MLB.

Mnemonic	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: buffer may not be word-aligned; buffer may be outside user address space; or buffer 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 is lost.
IE.ABO	-15	361	The link was aborted or disconnected (see disconnect and reject reason codes, Appendix A.)
IE.PRI	-16	360	The network is not accessed on this LUN.
IE.RSU	-17	357	Required system resources are not available.

(continued on next page)

Mnemonic	Decimal Value	Octal Value	Meaning
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.

Index

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

W

X

DECnet-RSX
Programmer's Reference Manual
AA-M098C-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

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TWO/E07
TEWKSBURY, MASSACHUSETTS 01876

DO NOT CUT FOLD HERE

CUT ALONG DOTTED LINE