

Professional™
300series

**PRO/DECnet Tool Kit
Programmer's Reference Manual**

Order No. AA-AV69A-TK

Developer's Tool Kit

digital
software

PRO/DECnet Tool Kit

Programmer's Reference Manual

Order No. AA-AV69A-TK

March 1984

This manual reviews software design conventions which are critical to the early stages of program development. It also details network programming calls used in the creation of PRO/DECnet applications.

SUPERSESSION/UPDATE INFORMATION: This is a new manual.

OPERATING SYSTEM AND VERSION: RSX-11M V4.1
RSX-11M-PLUS V2.1
VAX/VMS V3.4
P/OS V2.0

SOFTWARE VERSION: PRO/DECnet Tool Kit V1.0

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

Copyright © 1984 by Digital Equipment Corporation
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTI BUS	MASSBUS	RSTS
DEC	PDP	RSX
DECmate	P/OS	Tool Kit
DECsystem-10	PRO/BASIC	UNIBUS
DECSYSTEM-20	Professional	VAX
DECUS	PRO/FMS	VMS
DECwriter	PRO/RMS	VT
DIBOL	PROSE	Work Processor
Digital	Rainbow	

Distributed Systems Publications typeset this manual using DIGITAL's TMS-11 Text Management System.
--

CONTENTS

PREFACE

CHAPTER 1 GETTING STARTED WITH PRO/DECnet

1.1	Overview	1-1
1.2	PRO/DECnet Application Development Cycle	1-2
1.3	Required Libraries for Building PRO/DECnet Tasks	1-3
1.3.1	Programming in PASCAL	1-3
1.4	Creating PRO/DECnet Application and Object Installation Files	1-3
1.4.1	Formatting an Installation File	1-4
1.5	PRO/DECnet Application Installation Files	1-4
1.6	PRO/DECnet Object Installation Files	1-5
1.7	Combined PRO/DECnet Application/Object Installation Files	1-7
1.8	PRO/DECnet Object Description Files	1-8
1.8.1	OBJECT Command	1-8
1.8.2	RUN/INSTALL Command	1-10
1.8.3	RUN/REMOVE Command	1-10
1.9	PRO/DECnet Programming Considerations	1-12

CHAPTER 2 HIGH LEVEL LANGUAGE COMMUNICATION CALLS

2.1	Task Building	2-1
2.2	Assigning Logical Unit Numbers	2-2
2.3	Establishing an Active Network Task	2-2
2.4	Terminating Network Task Operations	2-2
2.5	Examining I/O Status Blocks	2-3
2.6	Access Control Information	2-3
2.7	Flow Control	2-3

2.8	Conventions Used in This Chapter.	2-4
2.9	Optional Arguments in High Level Language Calls.	2-5
2.9.1	Using Paired Optional Arguments	2-5
2.9.2	Using Single Optional Arguments	2-6
2.10	High Level Language Communication Calls	2-7
2.10.1	Common Argument Definitions.	2-8
2.10.2	ABTNT – Abort a Logical Link	2-10
2.10.3	ACCNT – Accept Logical Link Connect Request	2-13
2.10.4	BACC – Build Access Control Information Area.	2-16
2.10.5	BFMT0 – Build a Format 0 Destination Descriptor	2-20
2.10.6	BFMT1 – Build a Format 1 Destination Descriptor	2-23
2.10.7	CLSNT – End Network Task Operations	2-26
2.10.8	CONNT – Request Logical Link Connection.	2-28
2.10.9	DSCNT – Disconnect a Logical Link	2-31
2.10.10	GLNNT – Get Local Node Information	2-34
2.10.11	GNDNT – Get Network Data	2-37
2.10.12	OPNNT – Access the Network	2-43
2.10.13	RECNT – Receive Data over a Logical Link	2-47
2.10.14	REJNT – Reject Logical Link Connect Request	2-50
2.10.15	SNDNT – Send Data over a Logical Link	2-53
2.10.16	WAITNT – Suspend the Calling Task	2-56
2.10.17	XMINT – Send Interrupt Message	2-58

CHAPTER 3 MACRO-11 COMMUNICATION CALLS

3.1	MACRO Types	3-1
3.1.1	BUILD Type Macro.	3-2
3.1.2	EXECUTE Type Macro	3-3
3.1.3	STACK Type Macro	3-4
3.1.4	Macro Call Format Examples.	3-5
3.1.5	Macro Failures.	3-5
3.2	Using the Wait Option.	3-6
3.3	Using Asynchronous System Traps and Event Flags	3-6
3.4	Examining I/O Status Blocks	3-7
3.5	Assigning Logical Unit Numbers.	3-7
3.6	Establishing an Active Network Task	3-8
3.7	Access Control Information	3-8
3.8	Flow Control	3-8
3.9	Conventions Used in This Chapter.	3-9
3.10	MACRO-11 Communication Calls	3-10
3.10.1	Common Argument Definitions.	3-10
3.10.2	ABT\$ – Abort a Logical Link	3-12
3.10.3	ACC\$ – Accept Logical Link Connect Request	3-14
3.10.4	CLS\$ – End Network Task Operations	3-16
3.10.5	CON\$ – Request a Logical Link Connection	3-17
3.10.6	CONB\$\$ – Build Connect Block	3-20
3.10.7	DSC\$ – Disconnect a Logical Link	3-23
3.10.8	GLN\$ – Get Local Node Information	3-25
3.10.9	GND\$ – Get Network Data	3-28
3.10.10	OPN\$ – Access the Network	3-37
3.10.11	REC\$ – Receive Data over a Logical Link.	3-39
3.10.12	REJ\$ – Reject Logical Link Connect Request.	3-41
3.10.13	SND\$ – Send Data over a Logical Link	3-43
3.10.14	SPA\$ – Specify User AST Routine	3-45
3.10.15	XMI\$ – Send Interrupt Message	3-47

CHAPTER 4 DLX: DIRECT LINE ACCESS CONTROLLER

4.1	Special Considerations for Ethernet Users	4-2
4.2	DLX QIOs.	4-3
4.3	IO.XOP – Open the Ethernet Channel	4-4
4.4	IO.XSC – Set Characteristics	4-6
4.4.1	Setting up Protocol/Address Pairs	4-8
4.4.2	Setting up a Multicast Address.	4-9
4.5	IO.XTM – Transmit a Message on the Ethernet	4-10
4.5.1	Setting up the Ethernet Address	4-11
4.5.2	Setting the Protocol Type	4-11
4.6	IO.XRC – Receive a Message on the Ethernet	4-13
4.6.1	Optional Auxiliary Buffer for Receive Messages.	4-14
4.7	IO.XCL – Close the Ethernet Channel	4-16

CHAPTER 5 REMOTE FILE ACCESS

5.1	Introduction.	5-1
5.2	Using PRO/DECnet for Remote File Access	5-2
5.3	Formatting Remote Node Specifications	5-3
5.4	Remote Access Environments	5-4
5.5	Remote Access Pool Considerations.	5-4

APPENDIX A BASIC DECnet CONCEPTS

A.1	Task-to-task Communication	A-1
A.2	Establishing an Active Network Task	A-2
A.3	Building a Connect Block	A-2
A.3.1	Destination Descriptor	A-2
A.3.2	Source Descriptor	A-3
A.3.3	Access Control Information.	A-3
A.3.4	Optional Data Message	A-3
A.4	Assigning Logical Unit Numbers.	A-3
A.5	Establishing a Logical Link	A-4
A.6	Getting Data from the Network Data Queue	A-5
A.7	Accepting or Rejecting a Logical Link Connection Request.	A-5
A.8	Transmitting Data Messages over a Logical Link.	A-7
A.8.1	Sending Data Messages	A-7
A.8.2	Receiving Data Messages	A-7
A.9	Sending Interrupt Messages.	A-8
A.10	Using the I/O Status Block	A-8
A.11	Terminating Activity on a Logical Link	A-9
A.11.1	Disconnecting a Logical Link	A-9
A.11.2	Aborting a Logical Link	A-9
A.12	Closing the Network Connection.	A-10
A.13	DECnet Task-to-task Communication Calls	A-10

APPENDIX B DISCONNECT OR REJECT REASON CODES

APPENDIX C OBJECT TYPES

APPENDIX D MACRO-11 CONNECT BLOCK OFFSETS AND CODE DEFINITIONS

APPENDIX E ERROR/COMPLETION CODES FOR HIGH LEVEL LANGUAGES

APPENDIX F MACRO-11 ERROR/COMPLETION CODES

APPENDIX G SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

G.1	I/O Status Block Error Returns	G-1
G.2	Data Access Protocol (DAP) Error Messages	G-4
G.2.1	Maccode Field	G-5
G.2.2	Miccode Field	G-6

APPENDIX H TASK-TO-TASK PROGRAMMING EXAMPLES

H.1	FORTTRAN Programming Examples	H-1
H.1.1	FORTTRAN Transmit Program.	H-2
H.1.2	FORTTRAN Receive Program	H-4
H.2	COBOL Programming Examples.	H-6
H.2.1	COBOL Transmit Program	H-6
H.2.2	COBOL Receive Program	H-11
H.3	BASIC-PLUS-2 Programming Examples.	H-15
H.3.1	BASIC-PLUS-2 Transmit Program	H-15
H.3.2	BASIC-PLUS-2 Receive Program	H-17
H.4	PASCAL Programming Examples	H-19
H.4.1	PASCAL Transmit Program	H-19
H.4.2	PASCAL Receive Program	H-23
H.5	MACRO-11 Programming Examples.	H-26
H.5.1	MACRO-11 Transmit Program	H-26
H.5.2	MACRO-11 Receive Program	H-28
H.6	DLX QIO Programming Examples	H-31
H.6.1	DLX Transmit Program.	H-31
H.6.2	DLX Receive Program	H-40

FIGURES

1-1	Connecting PRO/DECnet Nodes to an Ethernet	1-1
3-1	Sample Connect Block Built by CONB\$\$	3-22
3-2	Sample Connect Block Returned by GND\$	3-36
A-1	Establishing a Logical Link	A-6

TABLES

2-1	High Level Language Communication Calls	2-7
2-2	BACC Connect Block Symbolic Offsets	2-19
2-3	BFMT0 Connect Block Symbolic Offsets.	2-22
2-4	BFMT1 Connect Block Symbolic Offsets.	2-25
2-5	Contents of Second Status Word Using GNDNT	2-38
3-1	MACRO-11 Communication Calls	3-10
3-2	CONB\$\$ Connect Block Symbolic Offsets	3-21
3-3	GND\$ Connect Block Symbolic Offsets	3-34
4-1	Summary of DLX Ethernet Calls.	4-3
A-1	DECnet Communication Calls Summary	A-11
G-1	First Word I/O Status Block Error Codes	G-2
G-2	NSP Error Codes	G-4
G-3	DAP Maccode Field Values	G-5
G-4	DAP Miccode Values for Use with Maccode Values of 2, 10, 11	G-6
G-5	DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7.	G-15
G-6	DAP Miccode Values for Use with Maccode Value 12	G-24

PREFACE

PRO/DECnet software, coupled with the appropriate hardware, allows a Professional 350 personal computer to connect to an Ethernet and to act as a DECnet Phase IV end node.

MANUAL OBJECTIVES

The *PRO/DECnet Tool Kit Programmer's Reference Manual* discusses software requirements for creating PRO/DECnet applications. It provides detailed information on the use of FORTRAN, COBOL, BASIC, PASCAL and MACRO-11 programming calls supported by PRO/DECnet.

It also assumes that you have a working knowledge of networking concepts.

INTENDED AUDIENCE

This manual is designed for application developers who are responsible for creating PRO/DECnet applications.

STRUCTURE OF THE MANUAL

This manual consists of 5 chapters and 8 appendices.

- ☐ **Chapter 1** discusses special software considerations when developing PRO/DECnet applications.
- ☐ **Chapter 2** details high level languages supported by PRO/DECnet. The programming calls for each language (FORTRAN, COBOL, BASIC, and PASCAL) are described in alphabetical order. Each description includes the call's format, argument(s) and associated error/completion status codes.

- ☐ **Chapter 3** discusses the use of MACRO-11 for PRO/DECnet task-to-task communication. The relevant MACRO-11 programming calls are described in a manner similar to the calls in Chapter 2.
- ☐ **Chapter 4** discusses the use of the Direct Line Access Controller (DLX) within an Ethernet networking environment. DLX calls are described in a manner similar to the calls in Chapters 2 and 3.
- ☐ **Chapter 5** discusses remote file access capabilities available to PRO/DECnet users.

NOTE

All references to FORTRAN imply FORTRAN-77, and all references to BASIC imply BASIC-PLUS-2 throughout this manual. In addition, all numbers supplied for the various arguments are decimal values unless otherwise noted.

This manual also contains 8 appendices:

- ☐ **Appendix A** discusses in greater detail DECnet terms and concepts which appear throughout this manual.
- ☐ **Appendix B** discusses error codes which are returned to the I/O status block for a rejected connect request or an aborted logical link.
- ☐ **Appendix C** lists values for specific network object types.
- ☐ **Appendix D** provides information on MACRO-11 connect block offsets used in network connects and accepts.
- ☐ **Appendix E** lists error/completion status codes for FORTRAN, COBOL, and BASIC programming calls.
- ☐ **Appendix F** lists error/completion status codes for MACRO-11 programming calls.
- ☐ **Appendix G** summarizes remote file access error/completion status codes.
- ☐ **Appendix H** contains programming examples for FORTRAN, COBOL, BASIC, PASCAL, MACRO-11 and DLX calls.

ASSOCIATED DOCUMENTS

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

PRO/RMS-11 Macro Programmer's Guide,
Order No. AA-P099A-TK

RSX-11M/M-PLUS Executive Reference Manual,
Order No. AA-L675A-TC

Tool Kit User's Guide,
Order No. AA-N617A-TK

VAX-11 Record Management Services Reference Manual,
Order No. AA-D031D-TE

GRAPHIC CONVENTIONS

UPPERCASE LETTERS	represent actual characters that you must enter as shown.
<i>lowercase italic type</i>	indicates variables whose value you must specify.
commas, periods, parentheses ()	must be included when shown as part of the call syntax. They are not documentation conventions.
square brackets []	enclose optional arguments. You must specify any argument not enclosed by brackets. Do not type the brackets when you code a call. They are documentation conventions and are not part of the call syntax.
angle brackets < > and commas	must be included when shown as part of a macro format. To omit an optional argument, do not specify a value for it but include its delineating comma only if there are no trailing arguments.
braces { }	enclose several keywords or arguments of which only one can be selected for a particular command or call. Do not include them as part of a command or call.

INTRODUCTION

DECnet is the name given to a family of software and hardware communications products that provide a network interface for Digital operating systems. The relationships between the various network components are governed by a set of standards called the Digital Network Architecture (DNA). Enhanced network capabilities and functions are incorporated into the different DECnet phases.

DECnet enables multiple computer systems to participate in communications and resource sharing within a specific network. The individual computer systems, called nodes, are connected by physical communications paths. Tasks that run on different nodes and exchange data are connected together by logical links. Logical links are temporary software information paths established between two communicating tasks in a DECnet network.

This manual describes the following network activities for Phase IV DECnet:

- ☐ Task-to-task communication – DECnet enables two programs to exchange data. These programs can reside in the same or in different nodes.
- ☐ Remote file access – DECnet provides both user and program access to files that reside on remote nodes. Remote file access subroutines allow you to transfer files between nodes, and to manipulate files residing on the remote nodes.
- ☐ Distributed data base access – Any network node can access information stored on any other network node. This feature enables you to use information stored in multiple data bases across the network.

CHAPTER 1

GETTING STARTED WITH PRO/DECnet

1.1 OVERVIEW

PRO/DECnet software allows Professional 350 computers to connect to other DECnet systems on the Ethernet. PRO/DECnet is an end node only implementation of the Phase IV Digital Network Architecture. It is compatible with other Phase III and Phase IV DECnet products.

Figure 1-1 shows several Professional 350 systems with DECNAs connected directly to an Ethernet network via H4000 Ethernet transceivers. This network topology also illustrates the compatibility of Professional 350s with remote VAX-11 host systems.

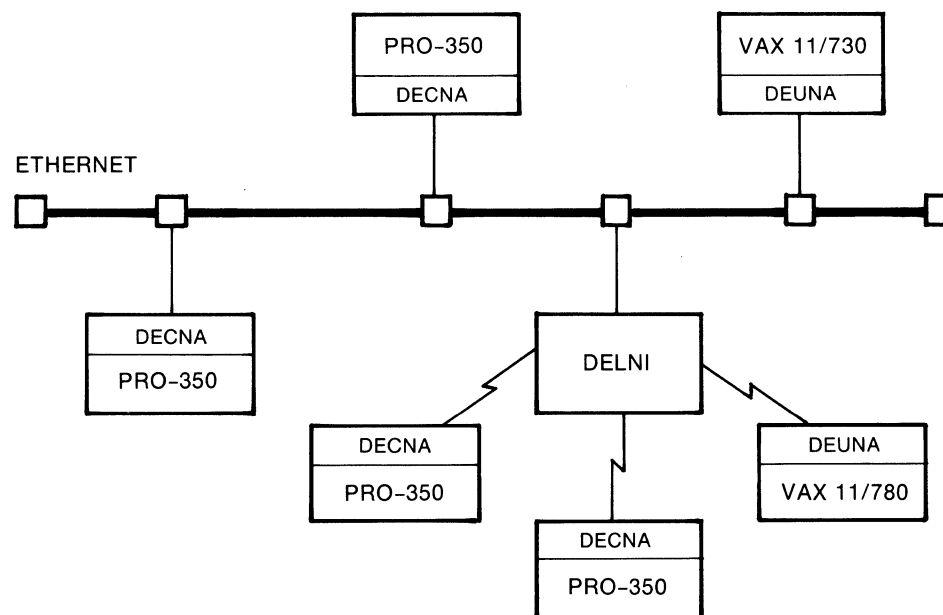


Figure 1-1
Connecting PRO/DECnet Nodes to an Ethernet

1-2 GETTING STARTED WITH PRO/DECnet

The PRO/DECnet software features:

- ☐ Supports multiple, simultaneous logical links between a Professional 350 and any other Phase III or Phase IV DECnet system.
- ☐ Supports task-to-task communication between a Professional 350 and any other Phase III or Phase IV DECnet system.
- ☐ Provides for high speed resource sharing within a local area network.
- ☐ Offers various network management and maintenance functions.
- ☐ Provides the transport facilities that permit programs utilizing RMS-11 V2.0 to access remote files.

NOTE

When the Professional Tool Kit resides on a Professional 350 computer, it is called the PRO/Tool Kit; when it resides on a host – either VAX/VMS or a PDP-11 running RSX-11M/M-PLUS – it is referred to as the Professional Host Tool Kit or, more commonly, the Host Tool Kit, for short.

1.2 PRO/DECnet APPLICATION DEVELOPMENT CYCLE

There are several steps involved in the development of a PRO/DECnet application. They include:

1. Writing the application and the application user interface
2. Compiling and task building the application (either on a host system or on a Professional)
3. Writing the application installation file
4. Transferring the application to the Professional (when developing programs on a Professional Host Tool Kit system)
5. Installing, executing and debugging the application
6. Copying the application to a diskette with the Application Diskette Builder

This chapter focuses on the libraries used by the Professional Application Builder (PAB) when you are building PRO/DECnet tasks – Step 2. It details the format and contents of PRO/DECnet application and object installation files – Step 3.

However, it does not discuss the complete applications development cycle. This information is documented in separate manuals or manual sets written specifically for the Host Tool Kit and the PRO/Tool Kit.

1.3 REQUIRED LIBRARIES FOR BUILDING PRO/DECnet TASKS

PRO/DECnet tasks are built with the Professional Application Builder (PAB). Before creating a PRO/DECnet task with PAB, you must create a command file (.CMD) and an overlay descriptor file (.ODL). In some cases, you can simply edit a command file automatically produced by certain Tool Kit language compilers.

PAB uses the command and descriptor files to define how libraries are referenced, and to specify special purpose buffers, logical unit numbers (LUNs) and event flags. (For instructions on how to invoke PAB, refer to the Tool Kit manual specific to your programming language.)

The following language libraries are required by PAB at task build time:

LB:[1,5]NETSUB.OLB

The NETSUB.OLB library contains a set of DECnet communication subroutines. PRO/DECnet applications written in FORTRAN, COBOL, and BASIC requiring task-to-task communication must be linked to this library.

LB:[1,5]NETLIB.MLB

The NETLIB.MLB library provides MACRO-11 macro definitions used by PRO/DECnet applications which require task-to-task communications.

LB:[1,5]NETDEF.PAS

The NETDEF.PAS library is a PASCAL file. It defines the various communication calls used by network programs written in PASCAL.

1.3.1 Programming In PASCAL

In order to use the DECnet communications calls for PASCAL programs, the PASCAL library must be referenced at the beginning of your source file. The command line should always be written as follows:

```
%INCLUDE 'LB:[1,5]NETDEF.PAS'
```

1.4 CREATING PRO/DECnet APPLICATION AND OBJECT INSTALLATION FILES

PRO/DECnet applications allow you to access network functions like network "phone" or "mail". They also enable you to access remote nodes for information sharing and program development.

In comparison, PRO/DECnet object tasks perform specific network services. They are automatically invoked whenever a request is issued from another node. Each user installed object task has a menu item. This menu line is required for the proper removal of the object from the system.

1-4 GETTING STARTED WITH PRO/DECnet

Some object tasks inform you of any remote requests which may require a personal response. For example, the phone listener tells you that someone is calling you from another Professional. Other object tasks place status messages on your system message board. For example, the mail listener tells you when new mail has been delivered to your Professional.

Sometimes, the application and object task are combined into one application. Whenever you install or remove a “combined” application, you are actually installing or removing both the application and the object. The PRO/DECnet Phone utility is an example of a combined package. Only the phone application is presented as an application menu item. The phone object simply runs as a background task when you select this particular application.

Application and object installation files identify all files and task images that are part of an application or object task. These files are needed for both the installation and removal of PRO/DECnet applications and objects.

You should use the disk/diskette services menu for installing and removing applications and objects. See the *Professional 300 User's Guide: Hard Disk System*, AA-N603A-TH, for details.

1.4.1 Formatting an Installation File

An application or object installation file must have an .INS file type and the following command format:

```
!This is the installation file
NAME "menu name"
FILE file name/value
INSTALL file name/value
RUN task name
```

You should refer to the *Tool Kit User's Guide* for additional details.

1.5 PRO/DECnet APPLICATION INSTALLATION FILES

The standard installation file is used for all PRO/DECnet applications which are not combined with PRO/DECnet objects. You should follow the syntax rules documented in the *Tool Kit User's Guide*.

Example

Here is a sample .INS file for a network virtual terminal application:

```

!
! Network Virtual Terminal
!
Name "Virtual Terminal"
!
! Specify all necessary files
!
File NVT.TSK/Delete
File NVTMENU.MNU/Delete
File NVTMESS.MSG/Delete
File NVTHELP.HLP/Delete
!
! Specify what tasks and libraries must be installed
!
Install NVT.TSK/Task
!
! Specify what to run when this application is selected
!
Run NVT

```

1.6 PRO/DECnet OBJECT INSTALLATION FILES

A PRO/DECnet object uses the standard application .INS file with some modifications. The differences include:

- ☐ a standard "application" task
- ☐ two executable tasks for each object task
- ☐ a description file for each object task

A network object task traditionally has no real application task to be installed or run whenever it is selected by the user. However, the Professional Operating System (P/OS) requires that an "application" task be supplied for each object task. These application tasks are used to install and later remove an object from the system. For this reason, PRO/DECnet supplies an application task called LB:[ZZDECNET]DAX.TSK. When this task is run, it tells you that the selected application is actually a PRO/DECnet object and should not be selected on its own.

Two executable tasks are provided for each object task. The DECnet application installation task, LB:[ZZDECNET]DAI.TSK, performs all needed functions to add the object to the network. The second task named the DECnet application removal task, LB:[ZZDECNET]DAR.TSK, performs the necessary steps to remove the task from the network.

Both tasks must be "executed" during the installation and removal of a PRO/DECnet object. A special "EXECUTE" directive defines the object to the network and must be issued with both procedures. The system requires that these tasks are executed in the proper order. The DAR task must be executed before any files are deleted during removal. The DAI task must be executed after all files are copied during installation. This specific requirement is illustrated in a later example.

As noted earlier, information must be provided about the object. The PRO/DECnet object description file, DECNET.ODS, contains the required description. This file is included on the installation diskette and must be copied to the application directory on the hard disk. The contents of DECNET.ODS are described in Section 1.8.

Example

Here is a sample .INS file for a network File Access Listener:

```
!
! Network File Access Listener
!
Name "File Access Listener"
!
! Remove the object from the network data base if
! performing the REMOVE operation. This must come
! before the "FILE" directives.
!
Execute [ZZDECNET]DAR.TSK/Rem
!
! Specify all necessary files to copy/delete
!
File FAL.TSK/Delete
File DECNET.ODS/Delete
!
! Insert the object in the network data base if
! performing an INSTALL operation. This must
! come after the "FILE" directives.
!
Execute [ZZDECNET]DAI.TSK/Ins
!
! Specify what tasks and libraries must be
! installed and run. Since the only task is
! an object, install and run DAX to inform
! the user that this application should not
! be selected.
!
! Note that installation of the FAL task is
! specified using the DECNET.ODS file.
!
Install [ZZDECNET]DAX.TSK/Task
Run DAX
```

1.7 COMBINED PRO/DECnet APPLICATION/OBJECT INSTALLATION FILES

The installation file for a combined PRO/DECnet application/object extracts information from both types of .INS files. The resultant file resembles an object .INS file with some exceptions. The application task [ZZDECNET]DAX.TSK should not be referenced in any command line. Instead, you should specify command lines for installing and running the appropriate application task.

Example

Here is a sample .INS file for the network Phone utility:

```

!
! Network Phone utility, application and listener
!
Name "PRO/DECnet Phone"
!
! Remove the object from the network data base if
! performing REMOVE operation. This must come
! before the "FILE" directives.
!
Execute [ZZDECNET]DAR.TSK/Rem
!
! Specify all necessary application files to copy/delete
!
File PHONE.TSK/Delete
File SETUP.TSK/Delete
File PHONE.MNU/Delete
File PHONE.HLP/Delete
File PHONE.MSG/Delete
!
! Specify all necessary object files to copy/delete
!
File PHONET.TSK/Delete
File SETNET.TSK/Delete
File DECNET.ODS/Delete
!
! Insert the object in the network data base if
! performing an INSTALL operation. This must
! come after the "FILE" directives.
!
Execute [ZZDECNET]DAI.TSK/Ins
!
! Specify what tasks and libraries must be
! installed and run. Since there is a real
! application, use that.
!
! Note that installation of the PHONET and SETNET
! tasks are specified using the DECNET.ODS file.
!
Install SETUP.TSK/Task
Install PHONE.TSK/Task
Run SETUP

```


1.8 PRO/DECnet OBJECT DESCRIPTION FILES

The DECNET.ODS file defines a PRO/DECnet object task. It is used by the DAI and DAR tasks during the installation and removal of applications. This file is included along with the .INS file in a common directory on the installation diskette. It must also be copied to the hard disk. This is done by specifying the file name with a FILE directive in an .INS file.

There are several commands and qualifying switches which can be placed in an object descriptor file. These commands can be abbreviated to three alphabetic characters.

NOTE

The file name argument, used in a DECNET.ODS command, adheres to the standard RSX-11M/M-PLUS conventions for file specifications. A file name contains up to 9 alphanumeric characters. It is followed by a 3 character alphanumeric file type.

An optional directory is also accepted in the file name. Only include it when a task is placed in a special directory by the .INS file. Otherwise, P/OS uses a default directory.

You can include comment lines anywhere in the command sequence. You begin a comment line with an exclamation point (!) and terminate it with a carriage return (RET). All text between this delimiter is a comment.

1.8.1 OBJECT Command

The OBJECT command indicates that the specified file is a PRO/DECnet object task file. This command requires the *file.ext* argument and the /TASK-NAME switch.

Format

```
OBJECT [dir]file.ext /TASKNAME=tsknam /COPIES=copnum
                                /NUMBER=objnum
                                /VERIFICATION=vertyp
```

```
{ INSPECT
  OFF
  ON }
```

Qualifying Switches**/TASKNAME=***tsknam*

specifies the task name to use when installing and defining the object. A task name consists of 1 to 6 alphanumeric characters. It can also contain periods (.) and dollar signs (\$).

/COPIES=*copnum*

specifies the number of copies of an object task to be simultaneously started by PRO/DECnet. This switch can only be used for numbered objects.

If specified, a new copy of an object is started up for each new connect request. The total number of copies can range from 1 to 8.

If you omit the /COPY switch, a single copy of the object task must be able to handle multiple connects.

NOTE

When you use the /TASKNAME switch for naming a "multi-copy object", the task name must contain 3 alphanumeric characters followed by 3 dollar signs (\$). You must also use the /NUMBER switch to specify an object type number from 1 to 255.

/NUMBER=*objnum*

specifies the object type number used by the source task when connecting to the object. This number can range from 0 to 255.

A named object is only referenced by its task name. As a result, its object type number is always 0. Do not include a /NUMBER switch when specifying a named object.

A numbered object has an object type number ranging from 1 to 255. Object type numbers 1 to 127 are reserved for DECnet-specific tasks. Numbers 128 to 255 are reserved for user-written tasks. (Refer to Chapters 2 and 3 for a discussion of named and numbered objects.)

`/VERIFICATION=vertyp`

specifies the degree to which you want access to a network object. There are three different options available for verifying incoming connect requests. If no switch is supplied, a default value of OFF is assumed.

- | | |
|---------|---|
| INSPECT | verifies the user ID and password supplied by the source program. The connect request is automatically forwarded to the object task regardless of the outcome. |
| OFF | allows connect requests to be passed along to the object without checking access authorization. This option is always used for named objects. |
| ON | verifies the user ID and password supplied by the source program. If there is no exact match, the connection request is rejected, and the object task does not receive the request. |

1.8.2 RUN/INSTALL Command

The RUN/INSTALL command causes the specified set-up task to run prior to installation of an application or at system startup. Before running an object task, certain initialization operations must be performed such as setting system parameters or defining system logical names.

This command requires the *file.ext* argument, and the /INSTALL and /TASK-NAME switches.

Format

`RUN [dir]file.ext/INSTALL/TASKNAME=tsknam`

Qualifying Switches

`/INSTALL`

directs the set-up task to run prior to installing an application.

`/TASKNAME=tsknam`

specifies the task name to use when running the set-up task. A task name consists of 1 to 6 alphanumeric characters. It can also contain periods (.) and dollar signs (\$).

1.8.3 RUN/REMOVE Command

The RUN/REMOVE command causes a tear-down task to run when an object application is removed from the system. This command enables required tear-down procedures to take place such as deleting logical names.

This command requires the *file.ext* argument, the /REMOVE and /TASKNAME switches.

Format

```
RUN [dir]file.ext/REMOVE/TASKNAME=tsknam
```

Qualifying Switches

```
/REMOVE
```

directs the task file to run prior to removing an application.

```
/TASKNAME=tsknam
```

specifies the task name to use when running the tear-down task. A task name consists of 1 to 6-alphanumeric characters. It can also contain periods (.) and dollar signs (\$).

Examples

Here is a sample DECNET.ODS file for a File Access Listener (FAL):

```
!
! Network File Access utility, object definition
!
! Define the FAL Listener as a PRO/DECnet object.
! This object has a specific object type number, and
! requires access verification.
!
! There is no set-up or tear-down task to run.
!
OBJECT [ZZDECNET]FAL.TSK/TAS=FAL$$$/NUM=17/VER=ON/COP=5
```

Here is a sample DECNET.ODS file for a network Phone Listener:

```
!
! Network Phone utility, object definition
!
! Define the Phone Listener as a PRO/DECnet object.
! This object is connected to by task name, so does not
! require an object number or verification definition.
!
OBJECT PHONET.TSK/TASKNAME=N.PHOL
!
! Run the Phone Listener set-up task at system startup
! time. This task defines a series of logical names to
! be used by the Phone Listener when it is automatically
! run by PRO/DECnet.
!
! No tear-down task is required when the application is
! removed. The logical names will be deleted when the
! system is next powered down.
!
RUN SETNET.TSK/INSTALL/TASKNAME=N.PHOS
```

1.9 PRO/DECnet PROGRAMMING CONSIDERATIONS

The following programming suggestions can assist you in writing and developing your PRO/DECnet applications:

1. Using a Trace Routine – You can add a trace routine to your PRO/DECnet program. This routine can assist you in collecting specific packets of data or allow you to set breakpoints at selected locations for examining specific instructions.
2. Debugging an Object Task – When you use a Professional 350 to debug an object task, you should select the PRO/Tool Kit from the Applications Menu. If the program stops executing, having an active PRO/Tool Kit will prevent your system from crashing.
3. Checking Software Compatibility – When creating PRO/DECnet programs, you should test them against a set of specific objectives. Is your task compatible with other tasks on your network? Can your task satisfy the requests of remote tasks?
4. Copying Files – Here are two cases when you should be careful about the use of the COPY command:
 - ☐ When copying either a Frame Development Tool (FDT) menu, a HELP file or a message file from a VAX/VMS node to your Professional 350 node, you **must** execute the COPY command from the Professional. The VMS COPY command does not use the block copy mode which is required by Professional systems for these transferring files. If the VMS command is used, you can expect a “record too long” error message displayed on your screen.
 - ☐ When you use P/OS DCL commands to access remote files, you must be aware of any differences in system conventions. If the file name syntax used on the remote system differs from the P/OS format, you should place quotes around the complete file name specification. The use of quotes is required for most VAX/VMS device and directory specification strings.

For example:

```
COPY VMSNOD"HARDY OLLIE"::"USER$: [HARDY, MEMO]FILE1, MEM" *.*
```

Here, a VMS file from device USER\$ and directory [HARDY.MEMO] was copied to a Professional system.

Error messages reflect a condition that prevented a command from executing properly. In most cases, the situation can be corrected and you can reissue the command. Follow these simple rules when writing a PRO/DECnet object task:

Problem 1 – Duplicating Object Task Names

When duplicated task names are found, PRO/DECnet displays an error message indicating the problem.

Solution

Assign task names to your application which are different from existing system task names. For OBJECT and RUN command task names, begin the names with the letter "N" followed by a period (.). For example, use "N.PHOS" for a phone set-up task or "N.PHOR" for a phone tear-down task. This solution does not apply to "multi-copy" objects. (See Section 1.8.1 for more details.)

Problem 2 – Duplicating Object Type Numbers

PRO/DECnet does not install an object task using a number already assigned to an installed task. An error message is displayed on the system message board.

Solution

Whenever possible, you should write your object task as a named object with object type number equal to 0.

NOTE

For named objects, the verification switch can only be set to "OFF". If you want to define the setting as either "ON" or "INSPECT", your object task must be numbered.

Problem 3 – DECNET.ODS File Errors

When an object task is installed, PRO/DECnet analyzes the DECNET.ODS file format. If any errors are found, an error message is displayed.

Solution

You should debug the file using the appropriate tools and repeat the installation procedure.

CHAPTER 2

HIGH LEVEL LANGUAGE COMMUNICATION CALLS

DECnet provides a set of subroutines for PRO/DECnet applications requiring task-to-task communication. These applications can be written in FORTRAN, COBOL, BASIC, and PASCAL. This chapter discusses the subroutine calls in alphabetical order. A description of each subroutine call includes its function, format, argument list, and associated error/completion codes. Unless otherwise noted, decimal values are supplied for all arguments discussed in this chapter.

If you encounter an unfamiliar DECnet concept or term, you should refer to Appendix A for more details.

2.1 TASK BUILDING

When you build PRO/DECnet tasks on a host system, you must invoke the Professional Application Builder (PAB). If you are building them on a Professional, you must invoke PAB by issuing a LINK command.

These types of tasks must also be linked to the library [1,5]NETSUB.OLB. To do this, add LB:[1,5]NETSUB/LB either to the task build command file or the overlay description file.

2.2 ASSIGNING LOGICAL UNIT NUMBERS

You can assign logical unit numbers (LUNs) for calls to the network (NS:) at task-build time or at run time. These specified LUNs must be assigned to NS: before they can be used in any network calls.

You may assign the LUN used to open the network (OPNNT) at task build time. In this case, you should not specify it in an OPNNT call. The symbol .MBXLU can define the LUN in a GBLDEF as shown below:

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

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

NOTE

After identifying the correct number of LUNs required for P/OS and language specific operations, you should include that number in the UNITS command for PAB. COBOL tasks cannot use LUN 1. It is a reserved number. In addition, PASCAL tasks should use LUNs from 25 through 40.

2.3 ESTABLISHING AN ACTIVE NETWORK TASK

The first DECnet call in your program must be an open call. An open call allows your task to access the network. You can use one of these forms:

- | | |
|--------|--|
| 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. This version of the call causes the issuing task to stop executing until the call has finished processing. |

Once an open call has been issued, you can establish several logical link connections for task-to-task communication.

2.4 TERMINATING NETWORK TASK OPERATIONS

A task can terminate network operations by issuing a close call in one of these forms:

- | | |
|--------|--|
| 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. This version of the call causes the issuing task to stop executing until the call has finished processing. |

2.5 EXAMINING I/O STATUS BLOCKS

All high level language calls, except for WAITNT, specify the I/O status block in their respective argument lists. This block contains completion status information on return from the completed call. The I/O status block takes these forms:

FORTTRAN: 1- or 2-word single-precision integer array

COBOL: 1- or 2-word elementary numeric data item

BASIC: 1- or 2-word integer array

PASCAL: 1- or 2-word single-precision integer array

BACC, BFMT0, and BFMT1 calls use one-word status blocks. For high level language tasks, a value of -1 or .TRUE. indicates that the call completed successfully; a value of 0 or .FALSE. indicates that the call contained an invalid argument.

All other calls use 2-word I/O status blocks. The first word contains an error/completion code for the call. The codes fall into three categories:

- ☐ A positive value means the successful completion of the call.
- ☐ A negative value means an improper execution of the call.
- ☐ A null value (0) indicates that the call has not finished processing.

When a call fails, you should examine the value of the code. A summary of code values is provided in Appendix E. Applicable error/completion codes can be found in each call description in this chapter.

The contents of the second status word vary with the call. Refer to the individual call descriptions for more information.

2.6 ACCESS CONTROL INFORMATION

Access control information is often required by a target system in order to prohibit unauthorized access to its resources. This information can consist of user ID, password, account numbers, device names and directory names for a target node. The specific requirements are described in the target system's user documentation. You can also define an alias for remote file access. An alias is a permanent "nickname" that you can assign to a node. It contains default access control information such as user ID, password and account number.

2.7 FLOW CONTROL

DECnet provides a flow control mechanism which prevents the overflow of available buffer space. It forces synchronization between sending and receiving tasks. When the flow control is ON, data is sent from the source task only after the target task has indicated adequate buffering capabilities, and has issued a receive call. (See Section 3.8 for a description of flow control with MACRO-11 tasks.)

2.8 CONVENTIONS USED IN THIS CHAPTER

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

UPPERCASE LETTERS	represent actual characters that you must enter as shown.
<i>lowercase italic type</i>	indicates variables whose value you must specify.
commas, periods, and parentheses ()	must be included when shown as part of the call syntax. They are not documentation conventions.
square brackets []	enclose optional arguments. You must specify any argument not enclosed by brackets. Do not type the brackets when you code a call. They are documentation conventions and are not part of the call syntax.

2.9 OPTIONAL ARGUMENTS IN HIGH LEVEL LANGUAGE CALLS

Many high level language calls contain both paired and single optional arguments. The following sections detail formatting conventions specific to each programming language.

2.9.1 Using Paired Optional Arguments

There are specific rules which you must follow when using paired optional arguments. The differences are discussed below:

Sample Argument List

arg1,arg2,arg3,[arg4,arg5],[arg6,arg7]

where *[arg4,arg5]* and *[arg6,arg7]* are paired optional arguments.

1. Paired optional arguments cannot be separated from each other. You must specify both or omit both from a call.
 - ☐ When paired optional arguments are omitted from FORTRAN and PASCAL calls, you must keep the arguments' positional commas as part of the list.

FORTRAN and PASCAL

arg1,arg2,arg3,,,[arg6,arg7]

- ☐ When paired optional arguments are omitted from COBOL and BASIC calls, you must specify 0 for each omitted argument.

COBOL

arg1,arg2,arg3,0,0,[arg6,arg7].

BASIC

arg1,arg2,arg3,0,0,[arg6,arg7]

2. An argument list can end with the last required argument contained in the string. You can omit any paired optional arguments which trail it. In the following example, *arg3* is the last required argument in a list of seven arguments. Paired optional arguments *[arg4,arg5]* and *[arg6,arg7]* are omitted from the example.

FORTRAN

arg1,arg2,arg3

2.9.2 Using Single Optional Arguments

Single optional arguments can also be omitted from a call's argument list. You should observe the same language conventions as previously discussed.

Sample Argument List

arg1, [*arg2*], *arg3*

where [*arg2*] is a single optional argument.

The following examples show the argument list without *arg2*.

FORTRAN

arg1.,*arg3*

COBOL

arg1,0,*arg3*.

BASIC

arg1,0,*arg3*

PASCAL

arg1.,*arg3*

2.10 HIGH LEVEL LANGUAGE COMMUNICATION CALLS

The following sections describe the high level language calls and provide you with specific guidelines. The calls are summarized in the table below:

Table 2-1
High Level Language Communication Calls

<i>Call</i>	<i>Function</i>
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 execution of a calling task.
XMINT	Send interrupt message over a logical link.

2.10.1 Common Argument Definitions

Commonly used arguments are defined in the following section. Each argument may have a general definition and three language-specific definitions. The information is not repeated with each intertask communication call.

outsize, outmessage

define optional user data sent with a specific call. One argument cannot be used or omitted without the other one.

outsize specifies the length of the optional message. The valid range is 1- to 16-bytes/characters.

outmessage specifies the array or string containing the outgoing user message.

EXCEPTION

To omit these arguments from the CONNT call in COBOL and BASIC, you must also omit the *insize* and *inmessage* arguments. Despite this, you can still use *insize* and *inmessage* and simply specify a null value (0) for *outsize* and *outmessage*.

status

specifies the array or string containing completion status information on return from a call. The *status* values for FORTRAN, COBOL, BASIC and PASCAL tasks are listed below:

FORTRAN

status(1) returns an error/completion code

status(2) contains 0 or a value which is DECnet call-dependent

COBOL

status(1) returns an error/completion code

status(2) contains 0 or a value which is DECnet call-dependent

BASIC

status%(0) returns an error/completion code

status%(1) contains 0 or a value which is DECnet call-dependent

PASCAL

<i>status</i> (1)	returns an error completion code
<i>status</i> (2)	contains 0 or a value which is DECnet call-dependent

EXCEPTION

For COBOL and BASIC tasks, the *status* argument cannot be omitted from a call. However, you can specify 0 to prevent status information from being returned from a call.

tgtblk

specifies the array or string where the access control information and destination descriptor are defined by the BACC and BFMT0 or BFMT1 calls. This array is passed to a target task by the CONNT call.

NOTE

A FORTRAN *tgtblk* array must start on an even byte (word) boundary.

ABTNT

Abort a Logical Link

2.10.2 ABTNT – Abort a Logical Link

The ABTNT call causes the immediate disconnection of a specified logical link. The associated LUN can be reassigned to another logical link. Along with the abort message, the issuing task can transmit an optional 1- to 16-bytes/characters message to the other task.

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%])
```

PASCAL

```
ABTNT[W] (lun,[status],[outsize,outmessage])
```

Arguments

lun

identifies the logical link to abort. If the task initiated the connection, specify the LUN used in the CONNT call. If the task accepted a connect request, specify the LUN used in the ACCNT call.

status

specifies the array which will contain completion status information on return from ABTNT. (See Section 2.10.1.)

outsize,outmessage

specify an optional message to be sent by a task. (See Section 2.10.1.)

Argument Data Type Summary**FORTTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>outsize</i>	integer variable or constant
<i>outmessage</i>	1- to 16-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>outsize%</i>	integer variable or constant
<i>outmessage\$</i>	1- to 16-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 2 No logical link established for the specified LUN.
- 9 The task is not a network task. OPNNT did not execute successfully.
- 13 An invalid buffer argument; the *outmessage* buffer is outside the user task's address space. For FORTRAN, it is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-Plus Executive Reference Manual*.

2-12 HIGH LEVEL LANGUAGE COMMUNICATION CALLS

Examples

FORTRAN

CALL ABTNTW (CONLUN,IOST,OUTSIZ,OUTMSG)

COBOL

CALL "ABTNTW" USING CONLUN,IOST,OUTSIZ,OUTMSG.

BASIC

CALL ABTNTW BY REF (CONLUN%,IOST%(),OUTSIZ%,OUTMSG\$)

PASCAL

ABTNTW (CONLUN,IOST,OUTSIZ,OUTMSG)

ACCNT

Accept Logical Link Connect Request

2.10.3 ACCNT – Accept Logical Link Connect Request

The ACCNT call establishes a logical link between the target task and the source task. Before calling ACCNT, the source task must call GNDNT and remove the connect request from the network data queue. The target task can return an optional 1- to 16-bytes/characters message to the source task.

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%()*],*mailbufr\$*,
[outsize%,outmessage\$])

PASCAL

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

Arguments

lun

assigns the logical unit number for the logical link. Use this LUN in succeeding RECNT, SNDNT, XMINT, ABTNT, and DSCNT calls.

status

specifies the array which will contain the completion status information on return from ACCNT. (See Section 2.10.1.)

mailbuf

specifies the connect block needed to establish the connection. The connect block was placed in *mailbut* by a preceding GNDNT call. (See Section 2.10.11.) In FORTRAN, the mail buffer must start on an even byte (word) boundary.

outside,outmessage

specify an optional message to be sent by a task. (See Section 2.10.1.)

Argument Data Type Summary**FORTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>mailbuf</i>	1- to 114-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>mailbuf</i>	1- to 114-element character string
<i>outsize</i>	integer variable or constant
<i>outmessage</i>	1- to 16-element character string

BASiC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>mailbuf\$</i>	1- to 114-element character string
<i>outsize%</i>	integer variable or constant
<i>outmessage\$</i>	1- to 16-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>mailbuf</i>	1- to 114-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 1 System resources needed for the logical link are not available.
- 3 The task that originally 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 issuing task is not a network task. OPNNT did not execute successfully.
- 13 An invalid buffer argument. Either the *mailbuf* or *outmessage* buffer is outside the user task address space. For FORTRAN, *mailbuf* is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-Plus Executive Reference Manual*.

Examples

FORTRAN

CALL ACCNTW (ACCLUN,IOST,MLBX,OUTSIZ,OUTMSG)

COBOL

CALL "ACCNTW" USING ACCLUN,IOST,MLBX,OUTSIZ,OUTMSG.

BASIC

CALL ACCNTW BY REF (ACCLUN%,IOST%(),MLBX\$,OUTSIZ%,
OUTMSG\$)

PASCAL

ACCNTW (ACCLUN,IOST,MLBX,OUTSIZ,OUTMSG)

BACC

Build Access Control Information Area

2.10.4 BACC – Build Access Control Information Area

The BACC call specifies the access control information for a connect block. For accessing files or performing privileged functions on other PRO/DECnet nodes, a user ID and a password are required access control information. Nodes with other system software may also require an account number.

You should use BACC only when the access control information is required for remote access of systems. Before calling BACC, the task must define a 72-element array or string in which the DECnet software will build the connect block.

If an alias contains the required access control information, the task need not call BACC. Instead, the program will use the access control information specified by the alias. The alias must be specified in a subsequent BFMT0 or BFMT1 call. (See Section 2.10.5 or 2.10.6.)

DECnet on the target system will check the validity of the access control information. If it checks out, the results are passed to the target task when the connect request is retrieved via the GNDNT call. (The GNDNT call only applies to RSX DECnet and PRO/DECnet systems.)

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$],[accnosz%,accno$])
```

PASCAL

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

Arguments*status*

specifies the array which will contain the completion status on return from BACC. When BACC completes successfully, *status* is set to .TRUE. for FORTRAN or to -1 for COBOL, BASIC and PASCAL. If there is an invalid BACC argument, *status* is set to .FALSE. for FORTRAN or to 0 for COBOL, BASIC and PASCAL. (See Section 2.10.1.)

tgtblk

specifies an array where the connect block is built. (See Section 2.10.1.)

usersz,user

specify the user ID. They are paired optional arguments. (See Section 2.9.1 for rules on omitting them from PRO/DECnet calls.)

usersz specifies the user ID length in bytes/characters.

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

passwdsz,passwd

specify the password associated with the user ID. They are paired optional arguments. (See Section 2.9.1 for rules on omitting them from PRO/DECnet calls.)

passwdsz specifies the password length in bytes/characters.

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

accnosz,accno

specify the account number. They are paired optional arguments. (See Section 2.9.1 for rules on omitting them from PRO/DECnet calls.)

accnosz specifies the account number length in bytes/characters.

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

Argument Data Type Summary**FORTRAN**

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>usersz</i>	1-word integer variable or constant
<i>user</i>	1- to 16-byte array
<i>passwdsz</i>	1-word integer variable or constant
<i>passwd</i>	1- to 8-byte array
<i>accnosz</i>	1-word integer variable or constant
<i>accno</i>	1- to 16-byte array

COBOL

<i>status</i>	integer variable
<i>tgtblk</i>	72-element character string
<i>usersz</i>	integer variable or constant
<i>user</i>	1- to 16-element character string
<i>passwdsz</i>	integer variable or constant
<i>passwd</i>	1- to 8-element character string
<i>accnosz</i>	integer variable or constant
<i>accno</i>	1- to 16-element character string

BASIC

<i>status%</i>	integer variable
<i>tgtblk\$</i>	72-element character string
<i>usersz%</i>	integer variable or constant
<i>user\$</i>	1- to 16-element character string
<i>passwdsz%</i>	integer variable or constant
<i>passwd\$</i>	1- to 8-element character string
<i>accnosz%</i>	integer variable or constant
<i>accno\$</i>	1- to 16-element character string

PASCAL

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>usersz</i>	1-word integer variable or constant
<i>user</i>	1- to 16-byte array
<i>passwdsz</i>	1-word integer variable or constant
<i>passwd</i>	1- to 8-byte array
<i>accnosz</i>	1-word integer variable or constant
<i>accno</i>	1- to 16-byte array

Table 2-2
BACC Connect Block Symbolic Offsets

<i>Length in bytes/characters</i>	<i>Destination Descriptor</i>
26	Built by BFMT0 or BFMT1 call 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

Examples

FORTRAN

```
CALL BACC (STAT,CONBLK,USRSIZ,USRNAM,PASSIZ,PASWRD,
          ACCSIZ,ACCNUM)
```

COBOL

```
CALL "BACC" USING STAT,CONBLK,USRSIZ,USRNAM,PASSIZ,
          PASWRD,ACCSIZ,ACCNUM.
```

BASIC

```
CALL BACC BY REF (STAT%,CONBLK$,USRSIZ%,USRNAM$,PASSIZ%,
          PASWRD$,ACCSIZ%,ACCNUM$)
```

PASCAL

```
BACC (STAT,CONBLK,USRSIZ,USRNAM,PASSIZ,PASWRD,ACCSIZ,
      ACCNUM)
```

BFMT0

Build a Format 0 Destination Descriptor

2.10.5 BFMT0 – Build a Format 0 Destination Descriptor

Use the BFMT0 call if the target task is installed as a numbered object. The BFMT0 call fills in the connect block with the destination node name and the target task's object type code. An object type code identifies a particular DECnet program by its function and not by its task name. See Appendix C for a list of object type codes.

Before calling BFMT0, the task must define a 72-element array or string which will contain the connect block. Specify its location in the *tgtblk* argument.

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*%)

PASCAL

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

Arguments

status

specifies the variable which will contain completion status information on return from BFMT0. When BFMT0 completes successfully, *status* is set to .TRUE. for FORTRAN or to -1 for COBOL, BASIC and PASCAL. For an invalid BFMT0 argument, *status* is set to .FALSE. for FORTRAN or to 0 for COBOL, BASIC and PASCAL. (See Section 2.10.1.)

tgtblk

specifies the starting location of the connect block. (See Section 2.10.1.)

ndsz

specifies the length of the target node name in bytes/characters.

ndname

specifies the location of the target node name. This name is either a node name or an alias.

objtype

specifies the target task's object type code. The valid range is 1 to 255. See Appendix C for a list of object type codes.

Argument Data Type Summary

FORTRAN

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>ndsz</i>	1-word integer variable or constant
<i>ndname</i>	1- to 6-byte array
<i>objtype</i>	1-word integer variable or constant

COBOL

<i>status</i>	integer variable
<i>tgtblk</i>	72-element character string
<i>ndsz</i>	integer variable or constant
<i>ndname</i>	1- to 6-element character string
<i>objtype</i>	integer variable or constant

BASIC

<i>status%</i>	integer variable
<i>tgtblk\$</i>	72-element character string
<i>ndsz%</i>	integer variable or constant
<i>ndname\$</i>	1- to 6-element character string
<i>objtype%</i>	integer variable or constant

PASCAL

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>ndsz</i>	1-word integer variable or constant
<i>ndname</i>	1- to 6-byte array
<i>objtype</i>	1-word integer variable or constant

Table 2-3
BFMT0 Connect Block Symbolic Offsets

<i>Length in decimal bytes/characters</i>	<i>Destination Descriptor</i>
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 a BACC subroutine

Examples

FORTRAN

CALL BFMT0 (STAT,CONBLK,NDLEN,NDNAM,OBJECT)

COBOL

CALL "BFMT0" USING STAT,CONBLK,NDLEN,NDNAM,OBJECT.

BASIC

CALL BFMT0 BY REF (STAT%,CONBLK\$,NDLEN%,NDNAM\$,OBJECT%)

PASCAL

BFMT0 (STAT,CONBLK,NDLEN,NDNAM,OBJECT)

BFMT1

**Build a Format 1
Destination Descriptor****2.10.6 BFMT1 – Build a Format 1 Destination Descriptor**

Use the BFMT1 call if the target task is installed as a named object. The BFMT1 call specifies the target node name and the target task name for the connect block. This call does not identify the target task by its function.

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%)
```

PASCAL

```
BFMT1 ([status],tgtblk,ndsz,ndname,objtype,namesz,name)
```

Arguments

status

specifies the variable which will contain completion status information on return from BFMT1. If the BFMT1 call completes successfully, *status* is set to .TRUE. for FORTRAN or to -1 for COBOL, BASIC and PASCAL. If there is an invalid BFMT1 argument, *status* is set to .FALSE. for FORTRAN or to 0 for COBOL, BASIC and PASCAL. (See Section 2.10.1.)

tgtblk

specifies the starting location of the connect block. (See Section 2.10.1.)

ndsz

specifies the length of the target node name in bytes/characters.

ndname

specifies the location of the target node name. This name is either a node name or an alias.

objtype

specifies the object type. It must be 0.

namesz

specifies the length of the target task name in bytes/characters.

name

specifies the location of the target task name.

Argument Data Type Summary

FORTRAN

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>ndsz</i>	1-word integer variable or constant
<i>ndname</i>	1- to 6-byte array
<i>objtype</i>	1-word integer variable or constant
<i>namesz</i>	1-word integer variable or constant
<i>name</i>	1- to 16-byte array

COBOL

<i>status</i>	integer variable
<i>tgtblk</i>	72-element character string
<i>ndsz</i>	integer variable or constant
<i>ndname</i>	1- to 6-element character string
<i>objtype</i>	integer variable or constant
<i>namesz</i>	integer variable or constant
<i>name</i>	1- to 16-element character string

BASIC

<i>status%</i>	integer variable
<i>tgtblk\$</i>	72-element character string
<i>ndsz%</i>	integer variable or constant
<i>ndname\$</i>	1- to 6-element character string
<i>objtype%</i>	integer variable or constant
<i>namesz%</i>	integer variable or constant
<i>name\$</i>	1- to 16-element character string

PASCAL

<i>status</i>	1-word integer variable
<i>tgtblk</i>	72-byte array
<i>ndsz</i>	1-word integer variable or constant
<i>ndname</i>	1- to 6-byte array
<i>objtype</i>	1-word integer variable or constant
<i>namesz</i>	1-word integer variable or constant
<i>name</i>	1- to 16-byte array

Table 2-4
BFMT1 Connect Block Symbolic Offsets

<i>Length in decimal bytes/characters</i>	<i>Destination Descriptor</i>
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 task name length (equal to or less than 16-bytes/characters)
16	Destination task name
Access Control	
46	Built by a BACC subroutine

Examples

FORTRAN

```
CALL BFMT1 (STAT,CONBLK,NDLEN,NDNAM,OBJTYP,TSKLEN,
            TSKNAM)
```

COBOL

```
CALL "BFMT1" USING STAT,CONBLK,NDLEN,NDNAM,OBJTYP,
            TSKLEN,TSKNAM.
```

BASIC

```
CALL BFMT1 BY REF (STAT%,CONBLK$,NDLEN%,NDNAM$,OBJTYP%,
            TSKLEN%,TSKNAM$)
```

PASCAL

```
BFMT1 (STAT,CONBLK,NDLEN,NDNAM,OBJTYP,TSKLEN,TSKNAM)
```

CLSNT

End Network Task Operations

2.10.7 CLSNT – End Network Task Operations

Before issuing CLSNT a task should disconnect all logical links and remove and process all messages on its network data queue. The CLSNT call disconnects the task from the network and aborts any active logical links. Any messages, except for connect requests, in the task's network data queue are discarded by this operation.

PRO/DECnet will save in a general delivery queue any connect requests received during CLSNT processing. After the task exits, the system software restarts it automatically. After restart, the task may issue an OPNNT call which establishes a network data queue. PRO/DECnet then places any connect requests on the task's network data queue.

Formats

FORTRAN

```
CALL CLSNT[W] [(status)]
```

COBOL

```
CALL "CLSNT[W]" [USING status].
```

BASIC

```
CALL CLSNT[W] [BY REF (status%)]
```

PASCAL

```
CLSNT[W] [(status)]
```

Arguments

status

specifies the array which will contain completion status information on return from CLSNT. (See Section 2.10.1.)

Argument Data Type Summary**FORTRAN**

status 2-word integer array

COBOL

status 2-element integer array

BASIC

status%() 2-element integer array

PASCAL

status 2-word integer array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 9 The task is not a network task. OPNNT did not execute successfully.
- 10 The network is not accessed on the specified LUN.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples**FORTRAN**

CALL CLSNTW (IOST)

COBOL

CALL "CLSNTW" USING IOST.

BASIC

CALL CLSNTW BY REF (IOST)

PASCAL

CLSNTW (IOST)

CONNT

Request Logical Link Connection

2.10.8 CONNT – Request Logical Link Connection

The CONNT call requests a logical link between a source task and a target task. Before a CONNT call is issued, you must build a connect block using the BACC and BFMT0 or BFMT1 calls. A target task can return an optional 1- to 16-bytes/characters message when it accepts or rejects the 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%])
```

PASCAL

```
CONNT[W] (lun,[status],tgtblk,
          [outsize,outmessage],[insize,inmessage])
```

Arguments

lun

assigns the logical unit number for the logical link. Use this LUN in any succeeding RECNT, SNDNT, XMINT, ABTNT, or DSCNT call.

status

specifies the array which will contain completion status information on return from CONNT. (See Section 2.10.1.)

tgtblk

specifies the array or string containing the connect block. (See Section 2.10.1.)

outsize,outmessage

define a buffer containing an optional outgoing message. They are paired optional arguments. (See Section 2.10.1 and note exception for BASIC and COBOL calls.)

insize,inmessage

specify a buffer containing an optional message sent by the target task. They are paired optional arguments. (See Section 2.10.1 and note exception for BASIC and COBOL calls.)

<i>insize</i>	specifies the length of the optional message in bytes/characters.
<i>inmessage</i>	specifies the 1- to 16-element/array or string which will store the incoming message.

Argument Data Type Summary**FORTTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>tgblk</i>	72-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array
<i>insize</i>	1-word integer variable or constant
<i>inmessage</i>	1- to 16-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>tgblk</i>	72-element character string
<i>outsize</i>	integer variable or constant
<i>outmessage</i>	1- to 16-element character string
<i>insize</i>	integer variable or constant
<i>inmessage</i>	1- to 16-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>tgblk\$</i>	72-element character string
<i>outsize%</i>	integer variable or constant
<i>outmessage\$</i>	1- to 16-element character string
<i>insize%</i>	integer variable or constant
<i>inmessage\$</i>	1- to 16-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>tgblk</i>	72-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array
<i>insize</i>	1-word integer variable or constant
<i>inmessage</i>	1- to 16-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Target task accepted the connection.
- 2 Target task accepted the connection and some returned data was lost.
- 1 Required system resources are not available for the logical link.
- 4 Target task rejected the connection and some returned data was lost.
- 5 The value of *outsize* is greater than 16 bytes.
- 7 Connection rejected by DECnet software on the target node. (See Appendix B.)
- 8 A logical link is already established on the specified LUN.
- 9 The source task was not a network task. CONNT did not execute properly.
- 12 The target task rejected the connection.
- 13 An invalid buffer argument. Either *tgtblk*, *inmessage* or *outmessage* buffer is outside the source task address space. For FORTRAN, the buffer is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

```
CALL CONNTW (CONLUN,IOST,CONBLK,,,INSIZ,INMSG)
```

COBOL

```
CALL "CONNTW" USING CONLUN,IOST,CONBLK,0,0,INSIZ,INMSG.
```

BASIC

```
CALL CONNTW BY REF (CONLUN%,IOST%(),CONBLK$,0,0,INSIZ%,INMSG$)
```

PASCAL

```
CONNTW (CONLUN,IOST,CONBLK,0,0,INSIZ,INMSG)
```

DSCNT**Disconnect a Logical Link****2.10.9 DSCNT – Disconnect a Logical Link**

The DSCNT call disconnects the logical link and frees the LUN associated with the logical link. Unlike ABTNT (see Section 2.10.2), all pending send data calls (SNDNT) will complete before you have a completely disconnected link.

The issuing task continues to receive outstanding data messages (RECNT calls) during processing time. Once the logical link disconnects, PRO/DECnet will reject any pending receive calls for the target task. These RECNT calls complete with an abort code (-3) in the I/O status block.

The task issuing DSCNT can send a 1- to 16-byte/character message to the target task.

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%])
```

PASCAL

```
DSCNT[W] (lun, [status], [outsize, outmessage])
```

Arguments

lun

specifies the logical unit number that you want to disconnect. If the task initiated the connection, specify the LUN used in the CONNT call. If the task accepted the connection, specify the LUN used in the ACCNT call.

status

specifies the array which will contain completion status information on return from DSCNT. (See Section 2.10.1.)

outsize, outmessage

specify a buffer containing the optional outgoing message. (See Section 2.10.1.)

Argument Data Type Summary**FORTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>outsize</i>	integer variable or constant
<i>outmessage</i>	1- to 16-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>outsize%</i>	integer variable or constant
<i>outmessage\$</i>	1- to 16-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 The call completed successfully.
- 1 No logical link established on the specified LUN.
- 5 The value of *outsize* is greater than 16 bytes.
- 9 The task is not a network task. CONNT did not execute properly.
- 10 The network is not accessed on the specified LUN.
- 13 An invalid buffer argument. The value for *outmessage* is outside the user task address space. For FORTRAN, the buffer is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL DSCNTW (CONLUN,IOST,OUTSIZ,OUTMSG)

COBOL

CALL "DSCNTW" USING CONLUN,IOST,OUTSIZ,OUTMSG.

BASIC

CALL DSCNTW BY REF (CONLUN%,IOST%(),OUTSIZ%,OUTMSG\$)

PASCAL

DSCNTW (CONLUN,IOST,OUTSIZ,OUTMSG)

GLNNT

Get Local Node Information

2.10.10 GLNNT – Get Local Node Information

The GLNNT call returns the local node name, node number and the default segment buffer size to be used on the logical link. The actual size is determined by comparing the default local segment size to the remote segment size as returned in the connect block. The smaller value becomes the value of the actual buffer size. For efficient message transmission, you should try to use a multiple of that number.

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$)
```

PASCAL

```
GLNNT[W] ([status],buflen,buf)
```

Arguments

status

specifies an array which will contain completion status information on return from GLNNT. (See Section 2.10.1.)

buflen

specifies the length of the array which will contain the received data. The returned data varies for each array length as summarized below:

Array length	Returned data
6-bytes/characters	local node name
8-bytes/characters	local node name, default segment buffer size
10-bytes/characters	local node name, node number, default segment buffer size

The first six bytes contain the local node name. The next two bytes contain the default segment buffer size. The last two bytes contain the local node number in the lower 10 bits and a value of 1 for the higher 6 bits.

buf

specifies the array or string containing the requested data.

Argument Data Type Summary

FORTRAN

<i>status</i>	2-word integer array
<i>buflen</i>	1-word integer variable or constant
<i>buf</i>	6-, 8- or 10-byte array

COBOL

<i>status</i>	2-element integer array
<i>buflen</i>	integer variable or constant
<i>buf</i>	6-, 8- or 10-element character string

BASIC

<i>status%()</i>	2-element integer array
<i>buflen%</i>	integer variable or constant
<i>buf\$</i>	6-, 8-, 10-element character string

PASCAL

<i>status</i>	2-word integer array
<i>buflen</i>	1-word integer variable or constant
<i>buf</i>	6-, 8-, or 10-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 4 Data overrun. The network data was longer than *buf*. The buffer is filled and any remaining data is lost.
- 9 The task is not a network task. OPNNT did not execute properly.
- 10 The network is not accessed on the specified LUN.
- 13 An invalid buffer argument. The value for *buf* is outside the user task address space. For FORTRAN, the buffer is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL GLNNTW (IOST,BUFSIZ,BUFFER)

COBOL

CALL "GLNNTW" USING IOST,BUFSIZ,BUFFER.

BASIC

CALL GLNNTW BY REF (IOST%(),BUFSIZ%,BUFFER\$)

PASCAL

GLNNTW (IOST,BUFSIZ,BUFFER)

GNDNT

Get Network Data

2.10.11 GNDNT – Get Network Data

The GNDNT call is used to retrieve information from the task's network data queue. There are five different types of messages which can be stored in a task's network data queue. Each message has an assigned type code as listed below:

Code	Message Type
1	Connect request
2	Interrupt
3	User disconnect
4	User abort
5	Network abort

A GNDNT call can check the queue or remove an entire message from the queue in one of these ways:

1. Retrieves the oldest message's type code, length, and associated LUN without removing it from the queue.

For message types 2 through 5, the associated LUN is returned to the high-order byte of the second status word. (Message type 1 has no associated LUN). Message type is returned to the *type* variable and message length is returned to the low-order byte of the second status word.

2. Removes the oldest message regardless of its type, length, or associated LUN. The message is placed in a mail buffer. The type code, message length, and associated LUN are returned in the status words as described in option 1.
3. Removes the first message for a particular type code, specified in *typmsk*, regardless of its associated LUN. The message is placed in the mail buffer. The type code, message length, and associated LUN are returned in the status words as described in option 1.
4. Removes the first message for a particular LUN, specified in *typmsk*, regardless of its type. The message is placed in the mail buffer. The type code, message length, and associated LUN are returned in the status words as described in option 1.
5. Removes the first message for a particular type for a specific LUN. Message type and LUN are specified in *typmsk*. The message is placed in the mail buffer. The type code, message length, and associated LUN are returned in the status words as described in option 1.

A GNDNT call must complete before you can issue a second GNDNT call. If not, the call will complete with an error (-14) in the low-order byte of the first word of the I/O status block.

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%])

PASCAL

GNDNT[W] ([status],type,[mailsz],[mailbuf],
[ltonly],[immed],[typmsk])

Arguments

status

specifies an array which will contain completion status information on return from GNDNT. See Section 2.10.1 for contents of first status word and Table 2-5 for contents of the second status word.

Table 2-5
Contents of Second Status Word Using GNDNT

<i>Message</i>		
<i>Type Code</i>	<i>Low-order byte</i>	<i>High-order byte</i>
1	Number of bytes/characters in connect request.	Access code: 2 = nonprivileged user 1 = privileged user 0 = no verification done -1 = verification failed
2	Number of bytes/characters in interrupt message.	Associated LUN
3	Number of bytes/characters in user disconnect message.	Associated LUN.
4	Number of bytes/characters in user abort message.	Associated LUN.
5	Reason for network abort (See Appendix B.)	Associated LUN.

type

specifies a variable which will contain the message type code on return from GNDNT.

mailsz

is required to retrieve a connect request message. It must also be used whenever *typmsk* or *ltonly* are specified in this call. It specifies the size of the task's mail buffer.

For a connect request, the connect block normally has a maximum of 98-bytes/characters without an optional data message. An optional data message can have 1- to 16-bytes/characters. If the two data items are added together, the resulting mail buffer cannot exceed a maximum length of 114-bytes/characters.

For message types 2 through 5, the mail buffer should be 1- to 16-bytes/characters to accommodate the optional data message.

mailbuf

is required if *ltonly* or *typmsk* is specified for retrieving a connect request message off the queue. It specifies a 1- to n-element array or string which will contain the message on return from GNDNT. In FORTRAN, the array must begin on an even byte (word) boundary.

ltonly

if *ltonly* is .TRUE. for FORTRAN or -1 for COBOL, BASIC or PASCAL, the oldest message on the network data queue is removed and placed in *mailbuf*. The type code is returned to the *type* variable. The message length is returned to the low-order byte of the second status word. The associated LUN is returned to the high-order byte of the second status word.

If *typmsk* is specified, either omit *ltonly* in FORTRAN, or specify 0 for *ltonly* in COBOL, BASIC and PASCAL.

immed

specifies the completion process for GNDNT. If there is at least one message on the queue, GNDNT will complete normally regardless of the value of *immed*.

If the queue is empty and the value of *immed* is .TRUE. in FORTRAN or -1 in COBOL, BASIC or PASCAL, GNDNT completes with error code (-6) in the low-order byte of the first status word.

If the queue is empty and the value of *immed* is .FALSE. in FORTRAN or 0 in COBOL, BASIC or PASCAL, GNDNT does not complete until the DECnet software places an incoming message on the queue.

typmsk

specifies the LUN and message type to be removed from the queue. (A message type of 0 implies any message type. A LUN at 0 implies any LUN.) The data is placed in *mailbuf* as follows:

Low-order byte	High-order byte
0 (any message type)	LUN
1 (connect request)	LUN or 0
2 (interrupt message)	LUN or 0
3 (user disconnect)	LUN or 0
4 (user abort)	LUN or 0
5 (network abort)	LUN or 0

Argument Data Type Summary**FORTRAN**

<i>status</i>	2-word integer array
<i>type</i>	1-word integer variable
<i>mailsz</i>	1-word integer variable or constant
<i>mailbuf</i>	1- to n-byte array
<i>lonly</i>	1-word integer variable
<i>immed</i>	1-word integer variable
<i>typmsk</i>	1-word integer variable or constant

COBOL

<i>status</i>	2-element integer array
<i>type</i>	integer variable
<i>mailsz</i>	integer variable or constant
<i>mailbuf</i>	1- to n-element character string
<i>lonly</i>	integer variable
<i>immed</i>	integer variable
<i>typmsk</i>	integer variable or constant

BASIC

<i>status%()</i>	2-element integer array
<i>type%</i>	integer variable
<i>mailsz%</i>	integer variable or constant
<i>mailbuf\$</i>	1- to n-element character string
<i>lonly%</i>	integer variable
<i>immed%</i>	integer variable
<i>typmsk%</i>	integer variable or constant

PASCAL

<i>status</i>	2-word integer array
<i>type</i>	1-word integer variable
<i>mailsz</i>	1-word integer variable or constant
<i>mailbuf</i>	1- to n-byte array
<i>lonly</i>	1-word integer variable
<i>immed</i>	1-word integer variable
<i>typmsk</i>	1-word integer variable or constant

Argument Usage Summary

Argument String	Usage
<i>lonly</i> and <i>typmsk</i>	use only one in a GNDNT call
<i>status</i> and <i>type</i>	retrieve the oldest message's code, length, and LUN without removing the entire message from the queue; omit the remaining arguments.
<i>status,type,lonly</i>	remove the oldest message from the queue regardless of its type, length or LUN; omit <i>typmsk</i> . A connect request message also requires <i>mailsz</i> and <i>mailbuf</i> . They are optional for interrupts, user disconnects, or aborts. You must specify them when a message length is greater than 0.
<i>typmsk,status,type</i>	remove a message for a desired type and/or LUN; omit <i>lonly</i> . A connect request message also requires <i>mailsz</i> and <i>mailbuf</i> . They are optional for interrupts, user disconnects, or aborts. You must specify them when a message length is greater than 0.

Error/Completion Codes

- 0 The call has not completed.
- 1 The call completed successfully.
- 2 The call completed successfully with some optional data loss. This occurs for a connect request when the source task includes an optional data message. This message is appended to the connect block.
- 4 Data overrun with some loss. The message removed from the queue was longer than *mailbuf*.
- 6 The task's network data queue was empty.
- 9 The task is not a network task. OPNNT did not execute successfully or a CLSNT call was issued during GNDNT processing. The latter can occur if the queue is empty and *immed* is *.FALSE.* for FORTRAN, or 0 for COBOL, BASIC or PASCAL.
- 10 The network is not accessed on the LUN specified in *typmsk*.
- 13 An invalid buffer argument. The value for *mailbuf* is outside the user task address space. For FORTRAN, the buffer is not word aligned.
- 14 A previously issued GNDNT is still being processed.
- 40 A directive error has occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL GNDNTW (IOST,MSGTYP)

CALL GNDNTW (IOST,MSGTYP,SIZE,BUFFR,.TRUE.)

CALL GNDNTW (IOST,MSGTYP,SIZE,BUFFR,,,MASK)

COBOL

CALL "GNDNTW" USING IOST,MSGTYP.

CALL "GNDNTW" USING IOST,MSGTYP,SIZE,BUFFR,-1.

CALL "GNDNTW" USING IOST,MSGTYP,SIZE,BUFFR,0,0,MASK.

BASIC

CALL GNDNTW BY REF (IOST%(),MSGTYP%)

CALL GNDNTW BY REF (IOST%(),MSGTYP%,SIZE%,BUFFR\$,-1%)

CALL GNDNTW BY REF (IOST%(),MSGTYP%,SIZE%,BUFFR\$,0,0,
MASK%)

PASCAL

GNDNTW (IOST,MSGTYP)

GNDNTW (IOST,MSGTYP,SIZE,BUFFR,-1)

GNDNTW (IOST,MSGTYP,SIZE,BUFFR,,,MASK)

OPNNT

Access the Network

2.10.12 OPNNT – Access the Network

The OPNNT call establishes the task as an active network task and creates the task's network data queue. The task 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%])
```

PASCAL

```
OPNNT[W] ([lun],[status],[mstat],
          [count],[lrp])
```

Arguments

lun

specifies the logical unit number for the task's network data queue. It can be omitted if a LUN was assigned to NS: during the task build operation by defining the symbol .MBXLU in a GBLDEF. (See Section 2.2). You must omit all trailing arguments when the *lun* argument is omitted in COBOL or BASIC. (See Section 2.9.)

status

specifies an array which will contain completion status information on return from OPNNT. (See Section 2.10.1.)

mstat

specifies an array which will contain information on the current status of the task's network data queue. The system continually updates the array as messages enter and leave the queue. Values returned in this array are:

mstat(1) number of messages on the queue

mstat(2) type of oldest message

- 1 (connect request)
- 2 (interrupt message)
- 3 (user disconnect message)
- 4 (user abort message)
- 5 (network abort message)

mstat(3) length of oldest message

count

assigns the maximum number of active logical links that a task can accept on the network. Once the number of active logical links equals the *count* value, the network will reject any pending connect requests. The valid range is 0 to 255, with 0 being the default value. A value of 0 implies that there is no limit to the number of links.

The *count* argument does not affect the number of logical links resulting from CONNT calls.

lrp

specifies the link recovery period. It defines the number of seconds that can elapse between a physical path failure and the disconnect of a logical link. The valid range is 0 through 32767. The default value is 0.

As long as the cooperating task remains connected and the physical path recovers before *lrp* elapses, the logical link will continue without any visible interruption. If the link does not recover in time, the system will abort the logical link.

If the recovery time is set to 0, and an alternate physical path cannot be immediately found, the system will immediately abort the logical link.

Argument Data Type Summary**FORTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>mstat</i>	3-word integer array
<i>count</i>	1-word integer variable or constant
<i>lrp</i>	1-word integer variable or constant

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>mstat</i>	3-element integer array
<i>count</i>	integer variable or constant
<i>lrp</i>	integer variable or constant

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>mstat%()</i>	3-element integer array
<i>count%</i>	integer variable or constant
<i>lrp%</i>	integer variable or constant

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>mstat</i>	3-word integer array
<i>count</i>	1-word integer variable or constant
<i>lrp</i>	1-word integer variable or constant

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 1 Required system resources were not available.
- 10 The network is being dismantled or the task has already issued a successful OPNNT call.
- 40 A directive error has occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL OPNNTW (OPNLUN,IOSTAT,MSTAT,MAXLNK,LNKR)

COBOL

CALL "OPNNTW" USING OPNLUN,IOSTAT,MSTAT,MAXLNK,LNKR.

BASIC

CALL OPNNTW BY REF (OPNLUN%,IOSTAT%(),MSTAT%,MAXLNK%,
LNKR%)

PASCAL

OPNNTW (OPNLUN,IOSTAT,MSTAT,MAXLNK,LNKR)

RECNT**Receive Data over a
Logical Link****2.10.13 RECNT – Receive Data over a Logical Link**

The RECNT call receives a data message over a logical link and stores it in a specified buffer. PRO/DECnet does not send a data message from one task to another until the intended receiver has issued a receive call. If the *indata* buffer is too small, the call completes with a data overrun condition (–4) in the first status word.

NOTE

The *indata* buffer should always align on an even byte boundary.

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*%)

PASCAL

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

Arguments

lun

specifies the logical unit number to be used for message transmission. If the task initiated the connection, specify the LUN used in the CONNT call. If the task accepted the connection, specify the LUN used in the ACCNT call.

status

specifies an array which will contain completion status information on return from RECNT. (See Section 2.10.1.)

insize

specifies the receive buffer length. The receive buffer can have a maximum length of 8128 bytes/characters.

indata

specifies the array or string which will contain the received message.

Argument Data Type Summary

FORTRAN

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>insize</i>	1-word integer variable or constant
<i>indata</i>	1- to 8128-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>insize</i>	integer variable or constant
<i>indata</i>	1- to 8128-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>insize%</i>	integer variable or constant
<i>indata\$</i>	1- to 8128-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>insize</i>	1-word integer variable or constant
<i>indata</i>	1- to 8128-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 2 No logical link established on the specified LUN.
- 3 The logical link was disconnected during an I/O operation.
- 4 Data overrun. More data was transmitted than requested.
- 9 Task is not a network task. OPNNT did not execute successfully.
- 13 An invalid buffer argument. The value for *indata* is outside the user task address space. For FORTRAN, the buffer is not word aligned, or *insize* specifies a value greater than 8128.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples**FORTRAN**

```
CALL RECNTW (CONLUN,IOSTAT,RECSIZ,RECBUF)
```

COBOL

```
CALL "RECNTW" USING CONLUN,IOSTAT,RECSIZ,RECBUF.
```

BASIC

```
CALL RECNTW BY REF (CONLUN%,IOSTAT%(),RECSIZ%,RECBUF$)
```

PASCAL

```
RECNTW (CONLUN,IOSTAT,RECSIZ,RECBUF)
```

REJNT

Reject Logical Link Connect Request

2.10.14 REJNT – Reject Logical Link Connect Request

The REJNT call rejects a logical link connect request. Along with a reject message, the task can send 1- to 16-bytes/characters of optional data to the requesting task.

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$,[outsize%,outmessage$])
```

PASCAL

```
REJNT[W] ([status],mailbuf,[outsize,outmessage])
```

Arguments

status

specifies the array which will contain completion status information on return from REJNT. (See Section 2.10.1.)

mailbuf

specifies the array or string containing the connect request message from the GNDNT call.

outsize,outmessage

specify an optional user message. (See Section 2.10.1.)

Argument Data Type Summary**FORTTRAN**

<i>status</i>	2-word integer array
<i>mailbuf</i>	1- to n-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

COBOL

<i>status</i>	2-element integer array
<i>mailbuf</i>	1- to n-element character string
<i>outsize</i>	integer variable or constant
<i>outmessage</i>	1- to 16-element character string

BASIC

<i>status%()</i>	2-element integer array
<i>mailbuf\$</i>	1- to n-element character string
<i>outsize%</i>	integer variable or constant
<i>outmessage\$</i>	1- to 16-element character string

PASCAL

<i>status</i>	2-word integer array
<i>mailbuf</i>	1- to n-byte array
<i>outsize</i>	1-word integer variable or constant
<i>outmessage</i>	1- to 16-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 3 The task requesting the connection has either aborted or requested a disconnect before the reject call could complete.
- 5 Either an invalid temporary link address in the connect block or the optional user data buffer exceeds 16-bytes/characters.
- 9 Task is not a network task. OPNNT did not execute successfully.
- 10 The network is not accessed on the specified LUN.
- 13 An invalid buffer argument. The value for *mailbuf* or *outmessage* is outside the user task address space. For FORTRAN, *mailbuf* is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL REJNTW (IOSTAT,BUFFR,OUTSIZ,OUTMSG)

COBOL

CALL "REJNTW" USING IOSTAT,BUFFR,OUTSIZ,OUTMSG.

BASIC

CALL REJNTW BY REF (IOSTAT%(),BUFFR\$,OUTSIZ%,OUTMSG\$)

PASCAL

REJNTW (IOSTAT,BUFFR,OUTSIZ,OUTMSG)

SNDNT

Send Data over a Logical Link

2.10.15 SNDNT – Send Data over a Logical Link

The SNDNT sends a data message over a particular logical link. The sending task does not actually transmit the message until the receiving task has issued a receive call. The sending task cannot reuse the *outdata* buffer specified in SNDNT until it receives a completion/error code from the system.

NOTE

The *outdata* buffer should always align on an even byte boundary.

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*%)

PASCAL

SNDNT[W] (*lun*,[*status*],*outsize*,*outdata*)

Arguments

lun

specifies the logical unit number to be used for message transmission. If the task initiated the connection, specify the LUN used in the CONNT call. If the task accepted the connect request, specify the LUN used in the ACCNT call.

status

specifies the array which will contain completion status information on return from SNDNT. (See Section 2.10.1.)

outsize

specifies the length of the outgoing message. The maximum length is 8128-bytes/characters.

outdata

specifies the array or string containing the outgoing data message.

Argument Data Type Summary**FORTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outdata</i>	1- to 8128-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>outsize</i>	integer variable or constant
<i>outdata</i>	1- to 8128-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>outsize%</i>	integer variable or constant
<i>outdata\$</i>	1- to 8128-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>outsize</i>	1-word integer variable or constant
<i>outdata</i>	1- to 8128-byte array

Error/Completion Codes

- 0 Call has not completed.
- 1 Call completed successfully.
- 2 No logical link established on the specified LUN.
- 3 The logical link was disconnected during I/O operation.
- 9 The task is not a network task. OPNNT did not execute successfully.
- 13 An invalid buffer argument. Either *outdata* is outside user task address space or the value of *outsize* exceeds 8128-bytes/characters.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL SNDNTW (CONLUN,IOSTAT,MSGLEN,MSGBUF)

COBOL

CALL "SNDNTW" USING CONLUN,IOSTAT,MSGLEN,MSGBUF.

BASIC

CALL SNDNTW BY REF (CONLUN%,IOSTAT%(),MSGLEN%,MSGBUF\$)

PASCAL

SNDNTW (CONLUN,IOSTAT,MSGLEN,MSGBUF)

WAITNT

Suspend the Calling Task

2.10.16 WAITNT – Suspend the Calling Task

The WAITNT call suspends task operation until a pending call, specified in an associated status block, has finished processing.

Formats

FORTRAN

CALL WAITNT (*index,status1,...,statusn*)

COBOL

CALL "WAITNT" USING *index,status1,...,statusn*.

BASIC

CALL WAITNT BY REF (*index%,status1%(),...,statusn%()*)

PASCAL

WAITNT (*index,status1,...,statusn*)

Arguments

index

specifies the variable which will contain the positional number of the status block associated with the completed call.

status1,...,statusn

specifies one or more status blocks associated with pending calls. For PASCAL calls, you can specify a maximum of 8 status blocks. (See Section 2.10.1.)

Argument Data Type Summary**FORTRAN**

<i>index</i>	1-word integer variable
<i>status1</i>	2-word integer array
<i>statusn</i>	2-word integer array

COBOL

<i>index</i>	integer variable
<i>status1</i>	2-element integer array
<i>statusn</i>	2-element integer array

BASIC

<i>index%</i>	integer variable
<i>status1%()</i>	2-element integer array
<i>statusn%()</i>	2-element integer array

PASCAL

<i>index</i>	1-word integer variable
<i>status1</i>	2-word integer array
<i>statusn</i>	2-word integer array

Examples**FORTRAN**

```
CALL WAITNT (INDEX, IOST1, IOST2, IOST3)
```

COBOL

```
CALL "WAITNT" USING INDEX, IOST1, IOST2, IOST3.
```

BASIC

```
CALL WAITNT BY REF (INDEX%, IOST1%(), IOST2%(), IOST3%())
```

PASCAL

```
WAITNT (INDEX, IOST1, IOST2, IOST3)
```

XMINT

Send Interrupt Message

2.10.17 XMINT – Send Interrupt Message

The XMINT call sends an interrupt message to a cooperating task over a logical link. The call completes when the source task is informed that the interrupt message was placed on the target task's network data queue.

The target task must remove the interrupt message from its network data queue before the source task can send another interrupt message over the same logical link.

NOTE

DECnet flow control does not apply to interrupt messages. These messages are delivered to a target task's buffer even if it has outstanding receive calls.

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*%)

PASCAL

XMINT[W] (*lun*,*[status]*,*intsize*,*intmsg*)

Arguments*lun*

specifies the logical unit number to be used for sending the interrupt message. If the task initiated the connection, specify the LUN that was used in the CONNT call. If the task accepted the connection, specify the LUN that was used in the ACCNT call.

status

specifies the array which will contain completion status information on return from XMINT. (See Section 2.10.1.)

intsize

specifies the length of the interrupt message. The length can range from 1- to 16-bytes/characters.

intmsg

specifies the array or string containing the interrupt message.

Argument Data Type Summary**FORTTRAN**

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>intsize</i>	1-word integer variable or constant
<i>intmsg</i>	1- to 16-byte array

COBOL

<i>lun</i>	integer variable or constant
<i>status</i>	2-element integer array
<i>intsize</i>	integer variable or constant
<i>intmsg</i>	1- to 16-element character string

BASIC

<i>lun%</i>	integer variable or constant
<i>status%()</i>	2-element integer array
<i>intsize%</i>	integer variable or constant
<i>intmsg\$</i>	1- to 16-element character string

PASCAL

<i>lun</i>	1-word integer variable or constant
<i>status</i>	2-word integer array
<i>intsize</i>	1-word integer variable or constant
<i>intmsg</i>	1- to 16-byte array

Error/Completion Codes

- 0 The call has not completed.
- 1 The call completed successfully.
- 2 No logical link established for 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 previously issued one was actually received by the remote task.
- 13 An invalid buffer argument. The value for *intmsg* is outside the user task address space. For FORTRAN, *intmsg* is not word aligned.
- 40 A directive error occurred. See the *RSX-11M/M-PLUS Executive Reference Manual*.

Examples

FORTRAN

CALL XMINTW (CONLUN,IOSTAT,MSGLEN,MSGBUF)

COBOL

CALL "XMINTW" USING CONLUN,IOSTAT,MSGLEN,MSGBUF.

BASIC

CALL XMINTW BY REF (CONLUN%,IOSTAT%(),MSGLEN%,MSGBUF\$)

PASCAL

XMINTW (CONLUN,IOSTAT,MSGLEN,MSGBUF)

CHAPTER 3

MACRO-11 COMMUNICATION CALLS

DECnet provides a library of MACRO-11 macros for PRO/DECnet applications that require task-to-task communication. The macro library [1,5]NETLIB.MLB is referenced by the Professional Macro Assembler (PMA) during assembly.

This chapter discusses the macro calls in alphabetical order. A section is devoted to each macro and its usage.

If you encounter any unfamiliar DECnet concept or term, refer to Appendix A for more details.

3.1 MACRO TYPES

There are three types of task-to-task communication macros:

- ☐ **BUILD type macro.** Creates an argument block at assembly time containing values for a specific network operation. It is used in conjunction with a DIR\$ directive or an EXECUTE type macro.
- ☐ **EXECUTE type macro.** References the argument block created by a BUILD type macro and executes the requested network function. An EXECUTE type macro gives you the option of overriding values specified by the BUILD type macro.
- ☐ **STACK type macro.** Creates an argument block on the processor stack and executes the requested function. The argument block contains values specified in the macro call.

3.1.1 BUILD Type Macro

An argument block is built containing values specified in the argument list. The argument block is then used by a DIR\$ directive or an EXECUTE type macro during task execution. A DIR\$ directive will generate less code than a corresponding EXECUTE type macro.

A DIR\$ directive pushes the address of the argument block onto the stack and issues an EMT 377. This action causes the executive to execute the macro. The format for a DIR\$ directive is:

DIR\$ *adr*,[*err*]

where

adr

specifies the address of the argument block as the source operand of a MOV instruction.

err

specifies the address of an optional, user-written error handling routine.

When a task uses both the BUILD and EXECUTE type macros, standard operations can be specified in the BUILD type macro and unique operations can be specified in the EXECUTE type macro. In this case, all arguments (required and optional) must be specified in a BUILD type macro argument block. If optional values are omitted, their positional commas must be included in the argument list.

For example, a send operation could have these standard functions specified in a BUILD type macro:

- ☐ the assigned LUN for a logical link
- ☐ an event flag number
- ☐ the location of the I/O status block containing completion status information upon return from a macro
- ☐ the starting address for a user-written AST routine

A corresponding EXECUTE type macro could specify these unique functions:

- ☐ the length of the outgoing message
- ☐ the location of the buffer containing the outgoing message

Format

label: XXX[W]\$ argument-list[,flag]

Arguments

label

specifies the location of the argument block.

XXX

specifies the name of a MACRO-11 call.

W

specifies that the issuing task will stop executing until the macro has finished processing. In order for the wait version to work properly, an event flag number (EFN) must also be specified in the argument list.

argument-list

defines one or more arguments separated by commas. Each specified value must be a valid argument for a .WORD or a .BYTE directive.

flag

specifies a call-dependent macro argument.

3.1.2 EXECUTE Type Macro

The EXECUTE type macro references arguments created by a BUILD type macro at task assembly time. An EXECUTE type macro enables you to override the argument block contained in the BUILD type macro. New arguments can either redefine values contained in the BUILD type macro or define missing values.

Format

XXX[W]\$E labelargument-list[,flag]

Arguments

XXX

specifies the name of a MACRO-11 call.

W

specifies that the issuing task will stop execution until the macro has finished processing. In order for the wait version to work properly, an event flag number (EFN) must be part of the argument list or previously specified in the BUILD type macro.

3-4 MACRO-11 COMMUNICATION CALLS

E

specifies an EXECUTE type macro call.

label

specifies the location of an argument block created by a BUILD macro. This argument must be valid for a .WORD directive.

argument-list

defines one or more arguments separated by commas. Each specified value must be a valid source operand for a MOV instruction.

flag

specifies a call-dependent macro argument.

3.1.3 STACK Type Macro

The STACK type macro creates the argument block on the processor stack and then executes the requested network function. All required arguments must be specified when you issue this macro. If not, the macro call will generate assembly errors.

Format

XXX[W]\$S *argument-list*[,*flag*]

Arguments

XXX

specifies the name of a MACRO-11 call.

W

specifies that the issuing task will stop executing until the macro has finished processing.

S

specifies a STACK type macro call.

argument-list

defines one or more arguments separated by commas. Each specified value must be a valid source operand for a MOV instruction.

flag

specifies a call-dependent macro argument.

3.1.4 Macro Call Format Examples

The following examples illustrate an interrupt message macro (XMI\$) coded in the three different call formats.

BUILD type macro call format

```
INTRPT: XMIW$ 1,FLAG,IOSTAT,,<MSGBUF,16.>
```

A BUILD type macro is assembled into an argument block referenced by INTRPT.

EXECUTE type macro call formats

```
XMIW$E INTRPT
```

The EXECUTE type macro references the argument block built by the corresponding BUILD type macro and executes the requested network operation.

```
XMIW$E INTRPT,,,,,<,#12.>
```

The EXECUTE type macro references the BUILD argument block, changes the buffer length from 16- to 12-bytes, and then executes the requested network operation.

```
XMIW$E IMSG,#1,#FLAG,#IOSTAT,,<#MSGBUF,#16.>
```

This EXECUTE type macro builds the required values into an empty argument block at the address label IMSG. It then executes the requested network operation. In this case, a BUILD type macro was not issued at the address label IMSG. However, a sufficient block of memory was reserved at that specific address.

STACK type macro call format

```
XMIW$$ #1,#FLAG,#IOSTAT,,<#MSGBUF,#16.>
```

This STACK type macro creates the argument block on the processor stack using specified argument values and then executes the requested network operation.

3.1.5 Macro Failures

At assembly time, the assembler expands each EXECUTE and STACK type macro to include a system directive. When the DIR\$ directive executes, it is possible that it can fail in the P/OS executive before DECnet can examine the request. If the system directive fails, the carry bit (C-bit) in the processor status word (PSW) is set when control returns to the instruction following the DIR\$ directive. If the C-bit is set, the directive status word (\$DSW) will contain an error code. After issuing a DIR\$ directive, the task should check the C-bit to insure that the P/OS executive accepted it.

3.2 USING THE WAIT OPTION

MACRO-11 calls can come in two forms - no wait and wait. The no wait version allows task operations to continue during the execution of a macro call. Here, one action does not depend on the completion of another action. The macro completes in an asynchronous manner.

The wait version causes an issuing task to stop running until the macro has finished processing. Once the macro completes, the issuing task proceeds at the instruction immediately following the macro call. All macros except for the CONB\$\$ macro allow you to use the wait option. The wait version is denoted by the letter W in the macro name and the inclusion of an event flag number in the macro call argument list. An event flag must always be specified for all macro calls that use the wait option. (See the following section for more details.)

3.3 USING ASYNCHRONOUS SYSTEM TRAPS AND EVENT FLAGS

An asynchronous system trap (AST) acts as a software interrupt and provides a task with a way of monitoring and reacting to certain events. When an event occurs, like a completed I/O operation, control passes to a user-written AST service routine. The starting address for the AST routine is specified in the macro call.

The corresponding address for the macro's status block is the first word on the processor stack. The AST routine must remove this word from the stack prior to its exit. ASTs can be specified in both wait and no wait type macros.

The task can use an event flag to recognize the same types of events. The event flag is cleared when the task issues a macro call. When the call completes, the system will set the event flag.

If P/OS rejects a call before DECnet can examine the request, the state of an event flag cannot be guaranteed. A task should therefore check the C-bit after issuing a request to insure that P/OS accepted it. (See Section 3.1.5)

NOTE

An event flag can be used in both wait and no wait macros as well as macros which do not specify AST routines. An event flag must always be specified for all macro calls that use the wait option ([W]). If not, the call completes in an asynchronous fashion as if the no wait option was specified with the call. FORTRAN, COBOL, BASIC and PASCAL programs use event flags 17 through 22. These flag numbers should not be assigned when a program also uses high level language task-to-task communication subroutine calls.

3.4 EXAMINING I/O STATUS BLOCKS

All MACRO-11 calls, except for CONB\$\$, can specify the I/O status block in their respective argument lists. The 2-word status block contains completion status information on return from the called macro. Byte 0 of word 0 will give you the following data:

- ☐ A zero indicates that a macro call has not finished processing.
- ☐ A positive value means the successful completion of a macro call.
- ☐ A negative value means improper execution of a macro call.

You can determine the reason for a macro's success or failure by comparing the value of byte 0, word 0 with a collection of symbolic values defined by the NSSYM\$ macro.

The symbolic values generally take one of these forms:

IS.xxx for a successful completion
IE.xxx for an error completion

where xxx is a specific value for the macro call. The remaining bytes in the 2-word status block are specific to each call. They are described in the following macro call descriptions.

You can invoke the NSSYM\$ macro by typing:

```
.MCALL NSSYM$
NSSYM$
```

3.5 ASSIGNING LOGICAL UNIT NUMBERS

All LUNs used in network macros must be assigned to the network device before you can issue any calls. You can assign LUNs by invoking the ALUN\$ system macro. For example, you can type:

```
ALUN$S #/un ,#"NS ,#0
```

3.6 ESTABLISHING AN ACTIVE NETWORK TASK

Before any task can exchange data using intertask communication calls, it must be a network task. As a result, the task must first issue the open macro call (OPN\$). The open macro call establishes the task as a network task and provides it with a network data queue. A task also requires a logical unit number (LUN) to identify its network data queue and act as a reference point for other macro calls.

The task's mailbox LUN can be defined during task execution by specifying it in the *lun* argument of the OPN\$ macro call. It can also be defined either during the assembly or the task build process. During the assembly process, the LUN can be locally defined in each source module by including the following:

```
.MBXLU==x
```

where the variable *x* is an integer greater than 0. The value of *x* must be the same in each source module.

In order to simplify your coding, you can also use a global definition in the task build command file as follows:

```
GLBDEF=.MBXLU:x
```

where the variable *x* is an integer greater than 0.

NOTE

You can define the network data queue LUN in only one place. You should use either .MBXLU or specify the LUN in the OPN\$ macro call. .MBXLU is referenced only if the *lun* argument is omitted from the OPN\$ macro call.

3.7 ACCESS CONTROL INFORMATION

Access control information is specified in the CONB\$\$ macro call. You should refer to the target system's user documentation for specific access requirements. See Section 2.6 for more information on access control information.

3.8 FLOW CONTROL

DECnet provides a flow control mechanism which prevents the overflow of available buffer space. It forces synchronization between sending and receiving tasks. When flow control is ON, data is sent from the source task only after the target task has indicated adequate buffering capabilities, and has issued a receive call.

MACRO-11 tasks also offer a special NOFLOW option which disables the flow control mechanism. The source task can disable its incoming flow control by specifying the NOFLOW flag in its connect request (CON\$). Likewise, the target task can disable its incoming flow control by specifying the NOFLOW flag in its accept call (ACC\$). This feature can be beneficial when a higher level of network performance is desired for a specific application. However, it should be used with caution.

When a source task disables flow control, data messages are sent from the target task regardless of buffering capabilities. This arrangement implies that a source task will manage its own resources and provide a buffer capable of handling the incoming data messages. If not, DECnet will be forced to discard some segments of the data messages. After a timeout, DECnet must request retransmission of each discarded data segment. This process could therefore significantly degrade network performance unless both tasks properly coordinate their resources.

3.9 CONVENTIONS USED IN THIS CHAPTER

The following conventions are used in the macro call descriptions:

UPPERCASE LETTERS	represent actual characters that must be specified as shown.
<i>lowercase italic type</i>	indicates variables for which a value must be supplied.
square brackets []	enclose optional data. You should specify all arguments not enclosed by square brackets. Do not include the brackets when coding a macro call.
braces { }	enclose several arguments of which only one can be selected for a particular call. Do not include the braces when coding a macro call.
commas and angle brackets < >	must be included when shown as part of a macro format. To omit an optional argument, do not specify a value for it but include its delineating comma. You can omit the trailing commas if there are no trailing arguments.

3.10 MACRO-11 COMMUNICATION CALLS

The following sections describe the use and format of the macros listed in Table 3-1. The macro calls appear in alphabetical order in this chapter.

Table 3-1
MACRO-11 Communication Calls

<i>Macro</i>	<i>Function</i>
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 a connect block for the CON\$ macro.
DSC\$	Disconnect a logical link.
GLN\$	Get local node information.
GND\$	Get data from the network data queue.
OPN\$	Access the network.
REC\$	Receive data over a logical link.
REJ\$	Reject a logical link connect request.
SND\$	Send data over a logical link.
SPA\$	Specify a user-written AST routine.
XMI\$	Send an interrupt message over a logical link.

3.10.1 Common Argument Definitions

Commonly used arguments are defined in the following section. Specific details about each argument are not repeated with each macro call.

status

specifies the address of the 2-word status block containing completion status information on return from a macro call. If specified, the status block can have one of the following values when the macro call completes:

Word 0: Byte 0 = Error/completion code
Byte 1 = 0

Word 1: Content depends on macro call

out, outlen

define optional user data to be sent with certain macro calls. One argument cannot be used or omitted without the other one.

out defines the starting address of a buffer containing optional user data.

outlen specifies the length of the optional outgoing message. The valid range is 1- to 16-bytes.

ABT\$

Abort a Logical Link

3.10.2 ABT\$ – Abort a Logical Link

The ABT\$ call immediately aborts all pending data transmission on a logical link. The link is disconnected and the task can subsequently reuse the associated LUN for another logical link. The task can send a 1- to 16-byte optional message with the abort notification to the other task.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

identifies the logical link to be aborted. If the task initiated the connection, specify the LUN that was used in the CON\$ call. If the task accepted the connect request, specify the LUN that was used in the ACC\$ call.

efn

specifies the event flag to be set when a ABT\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from ABT\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the ABT\$ call completes processing.

out, *outlen*

specify an optional outgoing user data message. (See Section 3.10.1.)

Error/Completion Codes

IS.SUC	The macro completed successfully.
IE.ABO	The specified logical link has already been disconnected.
IE.BAD	The data message exceeds 16-bytes.
IE.NLN	A logical link has not been established for the specified LUN.
IE.NNT	The task is not a network task. OPN\$ did not execute successfully.
IE.SPC	An invalid buffer argument. The message buffer <i>out</i> is outside the user task address space.

Examples

ABORT: ABTW\$ CONLUN,FLAG,IOSTAT,,<MSG,MSGLEN>

ABTW\$E ABORT

ABTW\$\$ #CONLUN,#FLAG,#IOSTAT,,<#MSG,#MSGLEN>

ACC\$

Accept Logical Link Connect Request

3.10.3 ACC\$ – Accept Logical Link Connect Request

The ACC\$ call establishes a logical link between two tasks. The task issuing the request is the target task. Before calling ACC\$, this task must call GND\$ to remove the connect request from its queue, and place the connect block in a mail buffer. The target task can send a 1- to 16-byte optional message with the accept message to the source task.

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

assigns the logical unit number for the logical link connection. Specify this LUN in succeeding REC\$, SND\$, XMI\$, ABT\$, and DSC\$ macro calls.

efn

specifies the event flag to be set when an ACC\$ call completes processing.

status

specifies the address of the two-word block containing completion status information on return from ACC\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the ACC\$ call completes processing.

mail

specifies the address of the connect block sent by the source task and retrieved by a GND\$ call. This address is identical to the one used in the GND\$ call. (See Section 3.10.9.) The connect block contains information needed to establish the connection. (See Table 3-2.)

mailen

specifies the length of the connect block. The length can range from 98- to 114-bytes. The length can exceed 98 bytes when the source task sends an optional data message. If omitted, the connect block defaults to 98 bytes.

out,outlen

define an optional outgoing user data message. (See Section 3.10.1.)

Flag

NOFLOW

disables flow control for incoming messages when specified. Flow control is the default condition.

Error/Completion Codes

IS.SUC	The macro completed successfully.
IE.ABO	The issuing task has aborted or has requested a disconnect prior to a completed connection.
IE.ALN	A logical link has already been established on the specified LUN.
IE.BAD	Either an invalid temporary link address in the connect block or the optional user data <i>outlen</i> exceeds 16-bytes.
IE.NNT	The issuing task is not a network task. OPN\$ did not execute successfully.
IE.RSU	No system resources available for a logical link.
IE.SPC	An invalid buffer argument. Either the connect block (<i>mail</i>) or the optional data buffer (<i>out</i>) is not word aligned, or is outside the user task address space.

Examples

```
ACCPT: ACCW$ CONLUN,FLAG,IOSTAT,,<MBUFFR>
```

```
ACCW$E ACCPT
```

```
ACCW$$ #CONLUN,#FLAG,#IOSTAT,,<#MBUFFR>
```

CLS\$

End Network Task Operations

3.10.4 CLS\$ – End Network Task Operations

Before issuing CLS\$, a task should disconnect all logical links and remove and process all messages on its network data queue. The CLS\$ call disconnects the task from the network and aborts any logical links. Any messages, except for connect requests, in the task's network data queue are then discarded by this operation.

PRO/DECnet will save in a general delivery queue any connect requests received during CLS\$ processing. When connect requests are found in the queue, DECnet will automatically restart the inactive task. After restart, the task issues an OPN\$ call which establishes a network data queue. PRO/DECnet then places any connect requests on the task's network data queue.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

efn

specifies the event flag to be set when a CLS\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from CLS\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after a CLS\$ call completes processing.

Error/Completion Codes

IS.SUC The macro completed successfully.

IE.NNT The task is not a network task. OPN\$ did not execute successfully.

IE.PRI The network is not accessed on the specified LUN.

Examples

CLOSE: CLSW\$ CONLUN,FLAG,IOSTAT

CLSW\$E CLOSE

CLSW\$\$ #CONLUN,#FLAG,#IOSTAT

CON\$**Request a Logical Link Connection****3.10.5 CON\$ – Request a Logical Link Connection**

The CON\$ call requests a logical link between two tasks. The target task and node are identified in the connect block.

A 1- to 16-byte optional message can be sent with the connect request to the target task.

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical unit number assigned to the logical link connection. Use this LUN in succeeding REC\$, SND\$, XMI\$, DSC\$, and ABT\$ calls.

efn

specifies the event flag to be set when a CON\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from CON\$. If specified, the block is set to one of these values:

Word 0: Byte 0 = Error/completion code
 Byte 1 = 0

Word 1: Byte 0 = Content depends on error completion code
 Byte 1 = 0

3-18 MACRO-11 COMMUNICATION CALLS

ast

specifies the starting address of an AST routine to be executed after the CON\$ call completes processing.

conbl

specifies the location of the connect block built by CONB\$\$.

conblen

specifies the length of the connect block. If this argument is omitted, the default length is 72-bytes.

out,outlen

define an optional outgoing message. (See Section 3.10.1.)

in,inlen

define the buffer which will receive the optional accept or reject message from the target task. They are paired optional arguments. You cannot use one without the other.

in is the address of the buffer.

inlen specifies the length of the incoming message. The valid range is 1-to 16-bytes.

Flag

NOFLOW

disables flow control for incoming messages when specified. Flow control is the default condition.

Error/Completion Codes

IS.SUC	The macro completed successfully.
IS.DAO	The macro completed successfully. The <i>in</i> block was not large enough for the optional user data sent by the target task.
IE.ALN	A logical link has already been established on the specified LUN.
IE.BAD	Either the <i>in</i> or <i>out</i> block exceeds 16-bytes, or the connect block length is too large.
IE.DAO	The target task issued a REJ\$ or REJNT call with optional user data which resulted in some data loss.
IE.NNT	The task is not a network task; OPN\$ did not execute successfully.
IE.NRJ	The network rejected the connection (see Appendix B).
IE.PRI	The local node is shutting down. No logical link can be established.
IE.RSU	No system resources are available for the logical link.
IE.SPC	An invalid buffer argument. Either the connect block (<i>conbl</i>) is not word aligned or the optional user data buffers (<i>in</i> or <i>out</i>) are outside the user task address space.
IE.URJ	The target task rejected the connection.

Examples

CONNECT: CONW\$ CONLUN,FLAG,IOSTAT,,CONBLK

CONW\$E CONNECT

CONW\$\$ #CONLUN,#FLAG,#IOSTAT,,#CONBLK

CONB\$\$

Build Connect Block

3.10.6 CONB\$\$ – Build Connect Block

This call builds a data area which is used as a connect block in a CON\$ macro call. A connect block must include the following data specified in the CONB\$\$ call:

- ☐ the target node name
- ☐ destination descriptor information (the target task object type, the descriptor format type, and, possibly, the target task name)
- ☐ access control information (the task's user ID and password on the target node, and if required, the task's account number). If an alias defines required access control information, it is unnecessary to define the information in the connect block.

Format

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

Arguments

NOTE

The following arguments can be defined in the CRBDF\$ call using a 72-byte block, N.RQL, with symbolic offsets. (See Table 3-2.)

The task can specify some values in N.RQL and then issue the CONB\$\$ call for only the remaining values. For specifying non-ASCII data for a field, you should create the field in N.RQL using the CRBDF\$ call, and omit the corresponding argument from the CONB\$\$ call.

node

specifies the target node name. It consists of 1- to 6-alphanumeric characters with at least one alphabetic character.

obj

specifies the target task's object type code. Object type codes range from 0 to 255. For a named object task, specify 0 as the object type code. For a numbered object task, specify a value from 1 to 255.

fmt

specifies the descriptor format type. If the value of *obj* is 0, specify 1 for the format type. Otherwise, specify 0.

descrip

specifies the target task name (1- to 16-ASCII characters) for a named object. Omit this argument for a numbered object.

NOTE

The *obj*, *fmt*, and *descrip* arguments comprise the destination descriptor.

rqid

specifies a legal user ID on the target node. It consists of 1- to 16- ASCII characters.

pass

specifies a legal password on the target node. It consists of 1- to 8-bytes. To specify a password using ASCII characters, precede each character with an apostrophe (') and separate them with commas. For example, the password PAS is specified as 'P','A','S'.

accno

specifies a legal account number on the target node. It consists of 1- to 16- ASCII characters.

NOTE

The *rqid*, *pass*, and *accno* arguments specify access control information.

Examples

```
CONB$$ BOSTON,0,1,<32W>,SMITH,<'T','O','M'>
```

```
CONB$$ BOSTON,129,0,,SMITH,<'T','O','M'>
```

Table 3-2
CONB\$\$ Connect Block Symbolic Offsets

<i>Symbolic Offset</i>	<i>Length in bytes</i>	<i>Contents</i>
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
		<i>Descriptor Field for Format 0</i>
	18	Not used
		<i>Descriptor Fields for Format 1</i>
N.RDEC	2	Destination task name length (equal to or less than 16-bytes)
N.RDE	16	Destination task name
ACCESS CONTROL INFORMATION		
N.RIDC	2	User ID length (equal to or less than 16-bytes)
N.RID	16	User ID
N.RPSC	2	Password length (equal to or less than 8-bytes)
N.RPS	8	Password
N.RACC	2	Account number length (equal to or less than 16-bytes)
N.RAC	16	Account number
N.RQL = 72		

* These symbolic offsets are guaranteed to be even (word aligned) when the connect block is built by the DECnet software. They can be defined using the CRBDF macro. (See Appendix D.)

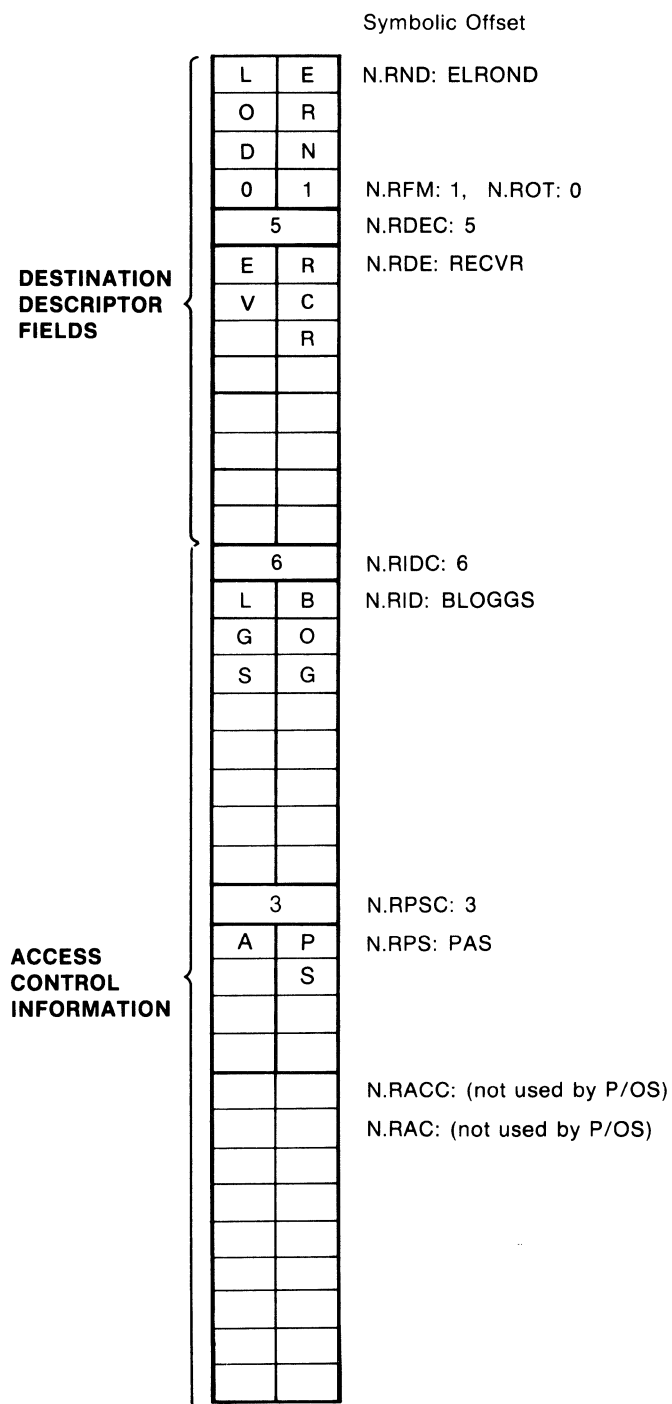


Figure 3-1
Sample Connect Block Built by CONB\$\$

Figure 3-1 illustrates a connect block which was built as a result of this call:

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

This connect block is directed to a task named RECVR on remote P/OS 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 the P/OS target system.

DSC\$**Disconnect a Logical Link****3.10.7 DSC\$ – Disconnect a Logical Link**

The DSC\$ call disconnects a logical link and frees its associated LUN. Unlike ABT\$ (see Section 3.10.2), DSC\$ causes all previously issued transmit calls (SND\$) to complete prior to disconnect.

When flow control is in effect, the task continues to receive outstanding data messages during the DSC\$ processing. When DSC\$ completes, the logical link is disconnected and PRO/DECnet rejects any pending incoming calls. These calls complete with an abort code (IE.ABO).

A task issuing DSC\$ can transmit a 1- to 16-byte optional message with the disconnect notification.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE macro calls.

lun

specifies the logical unit number for the link to be disconnected from the network. If the task initiated the connection, specify the LUN used in the CON\$ macro call. If the task accepted the connection, specify the LUN used in the ACC\$ macro call.

efn

specifies the event flag to be set when a DSC\$ call completes processing.

status

specifies the location of the 2-word block containing completion status information on return from DSC\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the DSC\$ call completes processing.

out,outlen

define optional outgoing user data. (See Section 3.10.1.)

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.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 accessible on the specified LUN.

IE.SPC An invalid buffer argument. The optional user data buffer *out* is outside the user task address space.

Examples

```
DISCON: DSCW$ CONLUN,FLAG,IOSTAT
```

```
DSCW$E DISCON
```

```
DSCW$$ #CONLUN,#FLAG,#IOSTAT
```

GLN\$

**Get Local Node
Information****3.10.8 GLN\$ – Get Local Node Information**

The GLN\$ call returns the local node name, the node number and the default segment buffer size to be used on the logical link. The actual segment size is determined by comparing the default local segment size to the remote segment size returned in the connect block. The smaller value becomes the value of the actual buffer size. For efficient message transmission, you should try to use a multiple of that number.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical unit number used to access the task's network data queue. This LUN is identical to the one specified in the OPN\$ call.

efn

specifies the event flag to be set when a GLN\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from GLN\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the GLN\$ call completes processing.

buf

specifies the block which will contain the requested data. The block must start on an even byte (word) boundary.

buflen

specifies the length of the array which will contain the received data. The returned data varies for each array length as summarized below:

Array length	Returned data
6-bytes/characters	local node name
8-bytes/characters	local node name, default segment buffer size
10-bytes/characters	local node name, node number, default segment buffer size

The first six bytes contain the local node name. The next two bytes contain the default segment buffer size. The last two bytes contain the local node number in the lower 10 bits and a value of 1 for the higher 6 bits.

Error/Completion Codes

IS.SUC	The macro completed successfully.
IE.DAO	Data overrun. The network data exceeded the specified buffer size which resulted in some data loss.
IE.NNT	The issuing task is not a network task. OPN\$ did not execute successfully.
IE.PRI	The network is not accessible on the specified LUN.
IE.SPC	An invalid buffer argument. The receiving buffer (<i>buf</i>) is outside the user task address space.

Byte 0 of word 1 in the I/O status block will contain the number of bytes transferred to *buf* if the call completed with IS.SUC or IE.DAO.

Examples

LOCNOD: GLNW\$ NTELUN,FLAG,IOSTAT,,BUFFR,BUFSIZ

GLNW\$E LOCNOD

GLNW\$\$ #NETLUN,#FLAG,#IOSTAT,,#BUFFR,#BUFSIZ

GND\$

Get Network Data

3.10.9 GND\$ – Get Network Data

The GND\$ call gets data from the task's network data queue and stores it in the specified mail buffer. If GND\$ completes successfully, word 0, byte 1 of the status block identifies which of the following messages have been stored:

- ☐ Type 1 – Connect request (NT.CON)
- ☐ Type 2 – Interrupt message (NT.INT)
- ☐ Type 3 – User disconnect notice (NT.DSC)
- ☐ Type 4 – User abort notice (NT.ABT)
- ☐ Type 5 – Network abort notice (NT.ABO)

You can use the SPA\$ macro to determine the number of data items in the network data queue. If you issue GND\$ or GND[W]\$ when the queue is empty, the call completes with an error code (IE.NDA).

GND\$ can receive data from a task's network data queue in these ways:

1. Retrieves the type code, message length and associated LUN for the oldest message without removing it from the queue. The call's status block returns the following information: The type code is returned to word 0, byte 1. The message length is returned to word 1, byte 0. For all messages except connect requests, the associated LUN is returned to word 1, byte 1.

You should specify these arguments:

...,*status*,...,NT.LON

2. Retrieves the length of the message specified in *mask* and returns it to word 1, byte 0 without removing the message from the queue. The message type and LUN are returned as described for option 1.

You should specify these arguments:

...,*status*,...,<,,*mask*>,NT.LON

NOTE

The *mask* argument can specify (1) a specific message type for a specific LUN, (2) a specific message type regardless of the associated LUN, or (3) a specific LUN regardless of the message type.

3. Removes the oldest message from the queue and places it in a mail buffer. The message type code, length, and LUN are returned as described for option 1.

You should specify these arguments:

...,*status*,...,<*mail*,*mlen*>

4. Removes the message specified by *mask* and places it in a mail buffer. The message type code, length, and LUN are returned as described in option 1. The *mask* argument can be specified as described in option 2.

You should specify these arguments:

...,*status*,...,<*mail*,*mlen*,*mask*>,NT.TYP

Formats

label: GND[W]\$ [*lun*],[*efn*],[*status*],[*ast*],
 $\left[\begin{array}{l} <mail,mlen> \\ <mail,mlen,mask>,NT.TYP \\ ,NT.LON \\ <,,mask>,NT.LON \end{array} \right\}$

GND[W]\$E *label*,[*lun*],[*efn*],[*status*],[*ast*],
 $\left[\begin{array}{l} <mail,mlen> \\ <mail,mlen,mask>,NT.TYP \\ ,NT.LON \\ <,,mask>,NT.LON \end{array} \right\}$

GND[W]\$S [*lun*],[*efn*],[*status*],[*ast*],
 $\left[\begin{array}{l} <mail,mlen> \\ <mail,mlen,mask>,NT.TYP \\ ,NT.LON \\ <,,mask>,NT.LON \end{array} \right\}$

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

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

efn

specifies the event flag to be set when a GND\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from GND\$. (See Section 3.10.1.) The content of the status block is summarized below:

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

Example 1**Status Word 0**

Byte 0 = IS.SUC or IS.DAO or IE.DAO

Byte 1 = NT.CON (Connect request)

Status Word 1

Byte 0 = Connect block length

Byte 1 = Access/privilege code:

VS.NPV = Nonprivileged user

VS.PRIV = Privileged user

VZ.NVD = No verification

VZ.NVD is returned to the status block whenever:

1) The target task was a named object with an object type code of 0. 2) The target task was a numbered object and did not require remote access verification.

VE.FAI = Verification failure

No account found in system file or password did not match the one found in the system file.

Example 2**Status Word 0**

Byte 0 = IS.SUC or IS.DAO or IE.DAO

Byte 1 = NT.INT (Interrupt)

NT.DSC (User disconnect)

NT.ABT (User abort)

Status Word 1

Byte 0 = Optional message length.

If 0, no message was received over LUN.

Byte 1 = LUN used either for the interrupt or disconnect message or the aborted logical link.

Example 3**Status Word 0**

Byte 0 = IS.SUC or IS.DAO or IE.DAO

Byte 1 = NT.ABO (Network abort)

Status Word 1

Byte 0 = Network abort code (see Appendix F)

Byte 1 = LUN used for the aborted logical link.

If GND\$ completes successfully and NT.LON is specified:

Status Word 0

Byte 0 = IS.SUC or IS.DAO or IE.DAO

Byte 1 = NT.XXX (Message type)

Status Word 1

Byte 0 = Message length

Byte 1 = Access/privilege code or the LUN associated with the message.

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

Status Word 0

Byte 0 = IE.XXX

Byte 1 = 0

Status Word 1

Byte 0 = 0

Byte 1 = 0

ast

specifies the starting address of an AST routine to be executed after the GND\$ call completes processing.

mail

specifies the address of the mail buffer which receives the data. For a connect request, *mail* is normally 98-bytes. If an optional user message was sent with the connect request, the maximum size for *mail* is 114-bytes.

For all other messages, *mail* should be 1- to 16-bytes in order to accommodate the optional data message sent with interrupt, user disconnect, and user abort messages. The value for *mail* must begin on a word boundary. See Table 3-3 for connect block symbolic offsets.

m/en

specifies the mail buffer size.

mask

specifies the message type (byte 0) and the LUN (byte 1) to be removed from the task's network data queue and placed in *mail* in one of these ways:

- ☐ If the specified message type equals 1, 2, 3, 4 or 5 and the associated LUN equals 0, then the first message for the specified message type, regardless of its associated LUN, is removed from the queue.
- ☐ If the LUN does not equal 0 and the message type equals 0, then the first message for a particular LUN, regardless of message type, is removed from the queue.
- ☐ If the specified message type and the LUN do not equal 0, then the first message for the specified message type and particular LUN is removed from the queue.

For example, to select the first disconnect message (NT.DSC) for LUN 3, code *mask* as follows: $3*256+NT.DSC$

Flags

NT.TYP

indicates that a specific message type and/or LUN has been requested in the *mask* argument. Always specify NT.TYP when *mask* is specified with *mail* and *m/en*.

NOTE

If NT.TYP is used in a BUILD type macro GND\$, it must also be included in the associated EXECUTE type GND\$.

NT.LON

indicates that the type and length of the message should be returned to the I/O status block. The message is not removed from the task's network data queue or placed in the mail buffer.

NOTE

If NT.LON is used in a BUILD type GND\$, it must also be included in the associated EXECUTE type GND\$.

Error/Completion Codes

IS.SUC	The macro completed successfully.
IS.DAO	The macro completed successfully but some returned optional data was lost in the process.
IE.DAO	Data overrun. The network data was longer than the mail buffer. As a result, some remaining data was lost in the transfer process.
IE.NDA	There is no data to return from the network data queue.
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	An invalid buffer argument. The buffer assigned to receive network data (<i>mail</i>) is not word aligned or is outside the user task address space.

Examples

```
GETNET: GNDW$ ,FLAG,IOSTAT,,<,,MASK>,NT.LON
```

```
GNDW$E GETNET,,,,,NT.LON
```

```
GNDW$$ ,#FLAG,#IOSTAT,,<,,#MASK>,#NT.LON
```

3-34 MACRO-11 COMMUNICATION CALLS

Table 3-3
GND\$ Connect Block Symbolic Offsets

<i>Symbolic Offset</i>	<i>Bytes</i>	<i>Contents</i>
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)
		DESTINATION DESCRIPTOR (20-byte total)
N.DFM	1	Destination descriptor format type: 0 or 1
N.DOT	1	Destination object type: 0-255
		<i>Descriptor Field for Format 0</i>
	18	Not used
		<i>Descriptor Fields for Format 1</i>
N.DDEC*	2	Destination task name length (equal to or less than 16-bytes)
N.DDE*	16	Destination task name
		SOURCE DESCRIPTOR (26-bytes total)
N.SND	6	Source node name (name of node requesting the connection, with trailing blanks)
N.SFM	1	Source descriptor format type (must be either format 0 or format 1)
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)
		<i>Descriptor Field for Format 0</i>
	18	Not used
		<i>Descriptor Fields for Format 1</i>
N.SDEC	2	Source task name length (equal to or less than 16-bytes)

* These symbolic offsets are guaranteed to be even (word aligned) when built by DECnet software. They can be defined using the CNBDF\$ macro. (See Appendix D.)

(continued on next page)

Table 3-3 (cont)
GND\$ Connect Block Symbolic Offsets

<i>Symbolic Offset</i>	<i>Bytes</i>	<i>Contents</i>
N.SDE	16	Source task name
		ACCESS CONTROL INFORMATION (46-bytes total)
		<i>If no verification is performed</i>
N.CIDC	2	User ID length (equal to or less than 16-bytes)
N.CID	16	User ID
N.CPSC	2	Password length (equal to or less than 8-bytes)
N.CPS	8	Password
N.CACC	2	Account number length (equal to or less than 16-bytes)
N.CAC	16	Account number
		<i>If verification is performed</i>
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
	40	Not used
		OPTIONAL DATA (18-bytes 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 N.CDA)		

* These symbolic offsets are guaranteed to be even (word aligned) when built by DECnet software. They can be defined using the CNBDF\$ macro. (See Appendix D.)

Figure 3-2 is an example of connect block information retrieved from the network data queue by a GND\$ call. This connect block was built by the CONB\$\$ call as shown in Figure 3-1.

3-36 MACRO-11 COMMUNICATION CALLS

		Symbolic Offset
DESTINATION DESCRIPTOR FIELDS		N.CTL: (not used by user)
	420	N.SEGZ: 420
	0 1	N.DFM: 1, N.DOT: 0
	5	N.DDEC: 5
	E R	N.DDE: RECVR
	V C	
SOURCE DESCRIPTOR FIELDS	O B	N.SND: BOSTON
	T S	
	N O	
	0 1	N.SFM: 1, N.SOT: 0
	5	N.SDEC: 5
	E S	N.SDE: SENDR
	D N	
ACCESS CONTROL: NO VERIFICATION	6	N.CIDC: 6
	L B	N.CID: BLOGGS
	G O	
	S G	
	3	N.CPSC: 3
	A P	N.CPS: PAS
		N.CACC: (not used by P/OS)
		N.CAC: (not used by P/OS)

Figure 3-2
Sample Connect Block Returned by GND\$

OPN\$**Access the Network****3.10.10 OPN\$ – Access the Network**

The OPN\$ call establishes the task as an active network task and creates the task's network data queue. The task must call OPN\$ before calling any other network macro.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

assigns a logical unit number to the task's network data queue. This argument can be omitted if a LUN was assigned to NS: by defining the symbol .MBXLU in the program or by using a GLBDEF during the task build operation.

Use the specified LUN in succeeding GND\$, SPA\$, GLN\$, REJ\$, and CLS\$ calls.

efn

specifies the event flag to be set when an OPN\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from OPN\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the GND\$ call completes processing.

links

assigns the maximum number of active logical links that are allowed for a task. Once the number of logical links, accepted via the ACC\$ macro, equals the *link* value, the network will automatically reject any pending connect requests. The valid range is 0 to 255 with 0 as the default value. A value of 0 implies that there is no limit to the number of logical links.

The *link* argument does not affect the number of logical links resulting from CON\$ calls.

lrp

specifies the link recovery period. It defines the number of seconds that can elapse between a physical path failure and a disconnect of a logical link. The valid range is 0 to 32767 with 0 as the default value.

As long as the cooperating task remains connected and the physical path recovers before *lrp* elapses, the logical link will continue without any visible interruptions. If the link does not recover in time, the system will abort the link.

If the recovery period is set to 0, and an alternate physical path cannot be immediately found, the system will immediately abort the logical link.

Error/Completion Codes

IS.SUC	The macro completed successfully.
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 unavailable.

Examples

```
OPNNET: OPNW$ 1,FLAG,IOSTAT
```

```
OPNW$E OPNNET
```

```
OPNW$$ #1,#FLAG,#IOSTAT
```

REC\$**Receive Data over a
Logical Link****3.10.11 REC\$ – Receive Data over a Logical Link**

The REC\$ call receives a data message over the logical link and stores it in a specified buffer. When flow control is in effect, a data message will not be sent between tasks until the intended receiver has issued a REC\$ call.

NOTE

The receive buffer *buf* should always align on an even byte boundary.

Formats

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

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

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

Arguments

label

specifies the location of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical link to be used for sending the message. If the task initiated the connection, specify the LUN used in the CON\$ call. If the task accepted the connection, specify the LUN used in the ACC\$ call.

efn

specifies the event flag to be set when a REC\$ call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from REC\$. The second word of the status block contains the actual number of received bytes. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed after the REC\$ call completes processing.

buf

specifies the address of the buffer which will contain the incoming data message.

buflen

specifies the length of the receive buffer in bytes. The valid range is 1- to 8128-bytes.

Error/Completion Codes

IS.SUC	The macro completed successfully.
IE.ABO	The logical link was disconnected during I/O operations.
IE.DAO	Data overrun. The incoming data was longer than the buffer. As a result, some data was lost in the transfer process.
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	An invalid buffer argument. Either the data buffer (<i>buf</i>) is outside the user task address space, or the buffer length (<i>buflen</i>) exceeds 8128 bytes.

Examples

```
RECDAT: REC 2,FLAG,IOSTAT,,<INBUF,BUFLEN>
```

```
REC$E RECDAT
```

```
REC$$ #2,#FLAG,#IOSTAT,,<#INBUF,#BUFLEN>
```

REJ\$

**Reject Logical Link
Connect Request****3.10.12 REJ\$ – Reject Logical Link Connect Request**

The REJ\$ call rejects a logical link request. The task can send 1- to 16-bytes of optional data with the reject message to the requesting task.

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]\$\$ [*lun*],[*efn*],[*status*],[*ast*],
<*mail*,[*mailen*],[*out*,*outlen*]>

Arguments

label

specifies the address of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical unit number assigned to the task's network data queue. Specify the LUN used in the OPN\$ call.

efn

specifies the event flag number to be set when a REJ\$ call completes processing.

status

specifies the address of the two-word block that contains completion status information on return from REJ\$. (See Section 3.10.1.)

ast

specifies the starting address of an AST routine to be executed when the REJ\$ call completes processing.

mail

specifies the address of the mail buffer that contains the connect block needed to reject the connect request. The connect block was placed in *mail* by the preceeding GND\$ call.

mailen

specifies the length of the connect block in bytes. The connect block can be 98- to 114-bytes. If *mailen* is omitted, the length defaults to 98 bytes.

out,outlen

define an optional outgoing message. (See Section 3.10.1.)

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.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	An invalid buffer argument. Either the connect block (<i>mail</i>) or the optional user data buffer (<i>out</i>) is outside the user task address space, or the connect block is not word aligned.

Examples

```
REJECT: REJW$ ,LUN,FLAG,IOSTAT,,<MBUF,MLEN>
```

```
REJW$E REJECT
```

```
REJW$$ ,#LUN,#FLAG,#IOSTAT,,<#MBUF,#MLEN>
```

SND\$

Send Data over a Logical Link

3.10.13 SND\$ – Send Data over a Logical Link

The SND\$ call sends a data message over a particular logical link. When flow control is in effect, the network does not actually transmit the message until the receiving task has issued a receive call. The sending task cannot reuse the message buffer specified in SND\$ until it receives an error/completion code from the network.

NOTE

The receive buffer *buf* should always align on an even byte boundary.

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 address of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical unit number to be used for message transmission. If the task initiated the connection, specify the LUN used in the CON\$ call. If the task accepted a connect request, specify the LUN used in ACC\$ call.

efn

specifies the event flag to be set when a SND\$ call completes processing.

status

specifies the address of the two-word block that contains completion status information on return from `SND$`. The second word of the status block contains the actual number of transmitted bytes. (See Section 3.10.1.)

3-44 MACRO-11 COMMUNICATION CALLS

ast

specifies the starting address of an AST routine to be executed when the SND\$ call completes processing.

buf

specifies the address of the buffer containing the outgoing data message.

buflen

specifies the buffer length in bytes. The buffer can be 1- to 8128-bytes.

Error/Completion Codes

- | | |
|--------|--|
| IS.SUC | The macro completed successfully. |
| IE.ABO | The logical link was disconnected during I/O operations. |
| 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 | An invalid buffer argument. Either the message data buffer (<i>buf</i>) is outside the user task address space, or the buffer length (<i>buflen</i>) exceeds 8128 bytes. |

Examples

```
SEND: SNDW$ 1,FLAG,IOSTAT,,<DATBUF,BUFLEN>
```

```
SNDW$E SEND
```

```
SNDW$$ #1,#FLAG,#IOSTAT,,<#DATBUF,#BUFLEN>
```

SPA\$

**Specify User AST
Routine****3.10.14 SPA\$ – Specify User AST Routine**

The SPA\$ macro defines an AST routine which will execute whenever data arrives in the network data queue. This AST routine will not be executed for any pre-call data already in queue. This data can be processed by a different AST routine specified in the *ast* argument of the SPA\$ call. The AST routine is executed after the SPA\$ macro completes processing. See the example at the end of this section.

Formats

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

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

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

Arguments*label*

specifies the address of the argument block for BUILD and/or EXECUTE type macro calls.

lun

specifies the logical unit number assigned to the task's network data queue. You should use the same LUN as specified in the OPN\$ call.

efn

specifies the event flag to be set when a SPA\$ call completes processing.

status

specifies the address of the 2-word block that contains completion status information on return from SPA\$. See definition in Section 3.10.1 and note this exception:

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

ast

specifies the starting address of an AST routine to be executed when the SPA\$ call completes processing.

addr

specifies the address of the user-written AST routine. The specified AST routine can be changed during execution by executing a different SPA\$ with a different starting address. It can be eliminated by using zero for the address. If this argument is omitted, no AST routine will be executed during task operation.

NOTE

SPA\$ executes whenever new data is placed in the network data queue. In this case, there is no additional data to be removed by the task from the stack.

Error/Completion Codes

- IS.SUC The macro completed successfully.
- 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.

Example

This example illustrates the use of the SPA\$ completion AST routine for processing network data.

MAIN PROGRAM CODE:

```
OPN$S...
SPA$S...,#CMPAST,<#SPAAST>
```

AST PROGRAM CODE:

```

      .ENABLE LSB
CMPAST:  MOV      (SP)+,IOSB      ;save SPA$ I/O status block addr
        MOV      R0,-(SP)        ;save R0 on stack
        MOV      IOSB,R0        ;set I/O status block addr
        CMPB     #IS.SUC,(R0)    ;successful?
        BNE      20$             ;if not equal, no - go exit AST
        MOV      2(R0),R0        ;set current network data count
        BEQ      20$             ;if zero, go exit AST
        BR       10$             ;continue with common code
SPAAST:  MOV      R0,-(SP)        ;save R0 on stack
        MOV      #1,R0           ;set network data count
10$:     GNDW$S   ...,#GNDSB      ;get network data item
        BCS      20$             ;if can't - go exit AST
        .
        .                       ;Process network data item
        .
        .
        SOB      R0,10$          ;loop till all items processed
20$:     MOV      (SP)+,R0        ;recover R0 from stack
        ASTX$S                   ;exit AST
        .DSABL  LSB
```

XMI\$**Send Interrupt Message****3.10.15 XMI\$ – Send Interrupt Message**

The XMI\$ call sends an interrupt message to a cooperating task over a logical link. The call completes when the source task is informed that the interrupt message was placed on the target task's network data queue. The target task must remove the interrupt message from the queue before the source task can issue another interrupt message over the same logical link.

Formats

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

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

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

Arguments

label

specifies the logical unit number to be used for sending the interrupt message. If the task initiated the connection, specify the LUN used in the CON\$ call. If the task accepted the connect request, specify the LUN used in the ACC\$ call.

efn

specifies the event flag set when the XMI\$ call completes processing.

status

specifies the address of the 2-word block that contains completion status information on return from the XMI\$ call. See definition in Section 3.10.1 and note this exception:

Word 1: Contains the number of bytes in the outgoing message.

ast

specifies the starting address of an AST routine when the XMI\$ call completes processing.

int

specifies the address of the buffer containing the outgoing interrupt message.

intlen

specifies the buffer length in bytes. The buffer can be 1- to 16-bytes.

Error/Completion Codes

IS.SUC	The interrupt message was transmitted successfully.
IE.ABO	The logical link was disconnected during I/O operations.
IE.BAD	The interrupt message exceeds 16 bytes.
IE.NLN	No logical link was established on the specified LUN.
IE.NNT	The issuing task is not a network task. OPN\$ did not execute successfully.
IE.SPC	An invalid buffer argument. The interrupt message buffer (<i>int</i>) is outside the user task address space.
IE.WLK	An interrupt message was transmitted before a previous interrupt message was actually received by the target task.

Examples

```
INTRPT: XMIW$ 1,FLAG,IOSTAT,,<INTBUF,BUFLEN>
```

```
XMIW$E INTRPT
```

```
XMIW$$ #1,#FLAG,#IOSTAT,,<#INTBUF,#BUFLEN>
```

CHAPTER 4

DLX: DIRECT LINE ACCESS CONTROLLER

The DLX interface allows you to send messages to DLX programs residing on other nodes without incurring the overhead of the higher level DECnet protocols. DLX programs benefit from the same low level networking software used for sending DECnet messages over the Ethernet.

A PRO/DECnet node can simultaneously run multiple DECnet and DLX tasks; each task could be communicating with different nodes. The DLX interface is automatically included in PRO/DECnet systems. To invoke DLX, you issue queued input/output (QIO) calls to the NX: device.

DLX can significantly improve network performance in terms of CPU utilization and response time. It allows you to build personalized user-level protocols which best suit your applications. By using DLX, your task loses services provided by the higher DECnet levels. Specifically, services such as resource management and checking for the occasional loss of packets must be provided in your user task to insure proper execution.

DLX programming requires a thorough knowledge of MACRO-11 assembly language and experience in writing real-time application programs. DLX does not support flow control for data transfer. As a result, you must write tasks that synchronize with each other before transferring data. If these tasks are not synchronized, the data can be lost during DLX communication. You must also provide your own error-handling routines. The DLX software informs you of any errors, but your task must include code for error recovery procedures.

NOTE

You must use the /PR switch to task build your DLX programs.

All DLX messages, both sent and received, are buffered in the same network buffer pool used by DECnet and other DLX tasks. Such sharing can effect their throughput performance. Depending on the DLX program, it may be advisable to increase the size and/or number of network buffers.

The DECnet test tool helps you isolate possible hardware problems. When a loopback test is initiated, a test message is sent from your node to a loopback connector or loopback software. Once the message reaches its destination, it is sent back to your node and compared to the original message. If you perform loopback tests on a DLX task containing multicast addresses, the task must be able to receive its own transmitted messages.

4.1 SPECIAL CONSIDERATIONS FOR ETHERNET USERS

The DECNA is a single device which can handle multiple simultaneous users. Externally, the DECNA appears as a single line point-to-point controller (for example, CNA-0). Internally, it is implemented like a multipoint device with each station representing an available port on the Ethernet.

All transmitted 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 addressing mode defines a unique address for a single node on any Ethernet. Multicast addressing mode defines a multidestination address of one or more nodes on a given Ethernet. Multicast addressing allows data to be transmitted only once to a number of nodes using a group address.

Each user can define unique protocol/address pairs for selecting specific messages for delivery. These pairs insure that messages will be delivered to the intended recipients.

The Ethernet offers three different routing methods:

- | | |
|-------------------------|--|
| 1. Exclusive
LF\$EXC | The user has exclusive use of the specific protocol and no other user may transmit or receive using this protocol. (DECnet routing uses this mode.) |
| 2. Default
LF\$DEF | The user should receive messages on this protocol that would otherwise be discarded because there was no protocol/address pair set up by this task or any other one. |
| 3. Normal | The user must specify the protocol/address pairs used for communications. |

NOTE

You should always select padding (LF\$PAD) in order to prefix a message with a 2-byte length field. This padding technique insures the minimum Ethernet size for proper transmission. If not, the field will be automatically added before the message is received at the destination. On receive, the length field will be used to indicate the amount of data present.

4.2 DLX QIOs

DLX requests conform to normal RSX-11 QIO standards. They also observe the standards for logical unit numbers (LUNs), event flags, I/O status blocks, asynchronous system traps (ASTs), and parameter lists. You can use any one of the three macro formats described in Section 3.1. The QIO WAIT option (specified as QIOW\$) can be used as versions of these macro calls.

The macros are defined in the DECnet macro library (NETLIB.MLB). 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 your programs. For example,

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

The DLX QIO calls and Ethernet functions are summarized in Table 4-1. Each call's arguments and completion status codes are described in Sections 4.3 through 4.7.

Table 4-1
Summary of DLX Ethernet Calls

<i>Code</i>	<i>Ethernet QIO</i>
IO.XOP	Open the Ethernet channel.
IO.XSC	Set Ethernet characteristics.
IO.XTM	Transmit a message on the Ethernet.
IO.XRC	Receive a message on the Ethernet.
IO.XCL	Close the Ethernet channel.

IO.XOP

Open the Ethernet Channel

4.3 IO.XOP – OPEN THE ETHERNET CHANNEL

This call opens a channel for direct message transfer and reception. It uses a LUN assigned to NX: and sets the appropriate protocol/address pairs.

To open the Ethernet device from DLX, issue this call using CNA-0 as the *device-id* string. DLX will scan the channel data base for an available channel and assign it for your use.

Format

QIO\$ IO.XOP,*lun*,[*efn*],[*status*],[*ast*],<*p1*,*p2*,*p3*>

Arguments

IO.XOP

is the function code that opens the channel.

lun

specifies a logical unit number already assigned to NX:. This LUN will be used in subsequent DLX calls.

efn

specifies an event flag set when the DLX call completes processing.

status

specifies the 2-word status block containing completion status information on return from the call. It is found in the low-order byte of the first word.

ast

specifies the starting address of the user-written AST routine.

p1

specifies the address of an ASCII string identifying the device upon which a channel will be opened for communication. The string is specified as *dev-ctl* where *dev* is the device mnemonic and *ctl* is the decimal value for the controller number. For PRO/DECnet, the only valid string is "CNA-0".

p2

specifies the length of the device identification field.

p3

specifies the time period which can elapse while the receiver waits for a message to be sent over the channel. The low-order byte of the word designates the receive timeout value as follows:

timeout = 0 for no receive timer.

timeout = $\langle n \rangle$

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 must equal 0.

If this timer elapses without data, the receive call will complete with an IE.TMO error.

Completion Status Codes

IS.SUC The channel has been opened successfully.
(1)

177646 You have either entered an invalid device identification format or
IE.NSF the specified device is not in the system.
(-26)

177736 The specified LUN is already in use.
IE.ALN (-34)

177757 The specified channel is already in use.
IE.RSU (-17)

177760 The specified channel is not available for use by DLX.
IE.PRI (-16)

IO.XSC

Set Characteristics

4.4 IO.XSC – SET CHARACTERISTICS

This call sets 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

specifies the logical unit number already used in the DLX open call.

efn

specifies an event flag number set when the DLX call completes processing.

status

specifies the address of the 2-word block containing completion status information on return from the call. The second word indicates how much of the characteristics buffer was processed during the call.

ast

specifies the starting address of a user-written AST routine.

p1

specifies the address of the characteristics buffer.

p2

specifies the length of the characteristics buffer in bytes.

Protocol Flag Codes

The following protocol flags are defined in EPMDF\$ (LF\$xxx). They are used when you set up a characteristics buffer. The flags and their symbolic offsets follow:

Protocol Flag	Octal Value	Meaning
LF\$EXC	1	Exclusive access protocol
LF\$DEF	2	Default user defined
LF\$PAD	4	Protocol requires padding
LF\$PRM	200	Permanent protocol

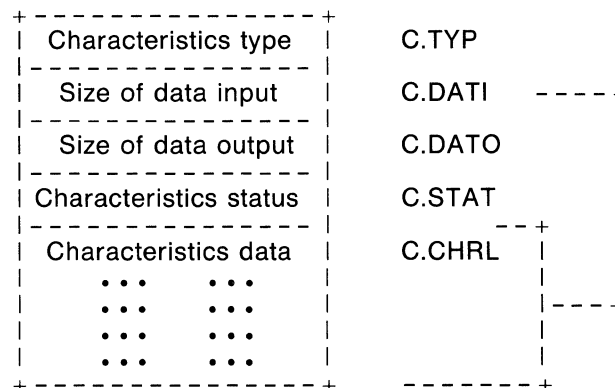
NOTE

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

Format of a Characteristics Buffer

The set characteristics buffer may contain multiple characteristics blocks. You can append these blocks into one characteristics buffer. Its length is specified in the *p2* argument of the IO.XSC call.

Each characteristics block has the following general format:



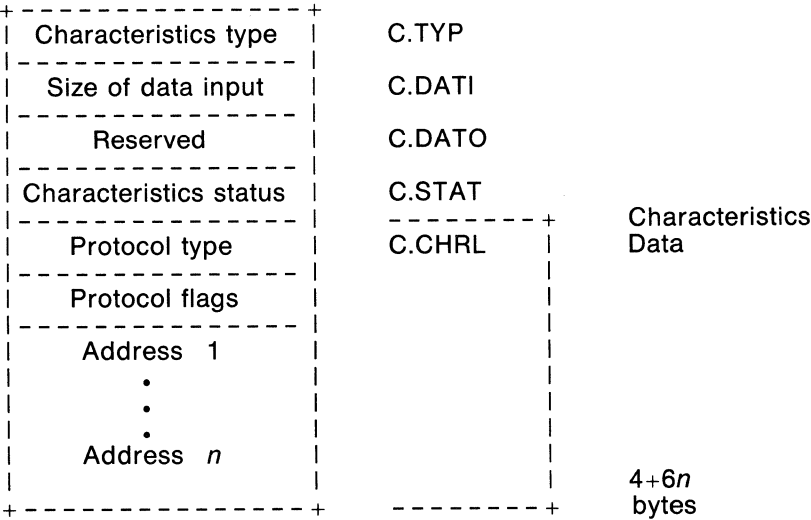
Entry	Symbolic Offset (Octal Value)	Meaning
C.TYP	0	Characteristics type code
C.DATI	2	Number of bytes of input data
C.DATO	4	Number of bytes of output data
C.STAT	6	I/O completion status return
C.CHRL	10	Minimum length of characteristics buffer

C.STAT Error Codes

100001	
CE.UDF	An undefined function
100003	
CE.RTL	Request was too large (too much data supplied).
100004	
CE.RTS	Request was too small (not enough data supplied).
100010	
CE.RES	Resource allocation failure

4.4.1 Setting up Protocol/Address Pairs

When the C.TYP entry is set to CC.DST (200), messages having the specified protocol type can be transmitted to and received from any address in the specified list. The characteristics data block (C.CHRL) is shown below:



Notes

1. The protocol type entry is a one word quantity which contains a user-defined 16-bit protocol value.
2. The protocol flags entry is a one word quantity which contains specific values as described in Section 4.4.
3. Each address entry is a three word quantity which contains the 48-bit Ethernet address.

C.STAT Error Codes

100011

CE.PCN Protocol usage conflict:

- A. Another user has exclusive access to this protocol.
- B. There is already a default user of this protocol. This request is attempting to set up a new default user.
- C. The padding status of this protocol does not match the requested one.

100012

CE.ACN The protocol/address pair is already in use.

100013

CE.IUN An illegal use of multicast addressing. One of the specified addresses is multicast.

4.4.2 Setting up a Multicast Address

When the C.TYP entry is set to CC.MCT (201), messages are received at the specified multicast address. The characteristics data block is shown below:

Characteristics type	C.TYP	
Size of data input	C.DATI	
Reserved	C.DATO	
Characteristics status	C.STAT	
Multicast address		Characteristics Data
		6 bytes

Note

- Each address entry is a three word quantity which contains the 48-bit Ethernet address.

C.STAT Error Codes

100007

CE.MCE The specified multicast address is already enabled.

100014

CE.NMA The specified address is not a multicast address.

IO.XTM

Transmit a Message on the Ethernet

4.5 IO.XTM – TRANSMIT A MESSAGE ON THE ETHERNET

When you transmit a message on the Ethernet, you must specify its destination multicast or physical address and protocol type. This information is specified in the *p3* and *p4* arguments of an auxiliary characteristics buffer.

The auxiliary characteristics buffer has the same format as the set characteristics buffer. See Section 4.6 for more information.

Format

QIO\$ IO.XTM,*lun*,[*efn*],[*status*],[*ast*],<*p1*,*p2*,*p3*,*p4*>

Arguments

IO.XTM

is the function code for transmitting a message.

lun

specifies the logical unit number already used in the DLX set characteristics call.

efn

specifies an event flag number set when the IO.XTM call completes processing.

status

specifies the address of the 2-word status block containing completion status information on return from the call. This information is contained in the low-order byte of the first word.

ast

specifies the starting address of a user-written AST routine.

p1

specifies the address of the user buffer containing the outgoing message.

p2

specifies the length of the outgoing message.

p3

specifies the address of the auxiliary characteristics buffer containing the multicast or physical addresses.

p4

specifies the length of the auxiliary characteristics buffer.

4.5.1 Setting up the Ethernet Address

You must define the protocol type by setting the C.TYP entry to CC.ADR (100) before messages can be successfully transmitted over the Ethernet channel. The individual characteristics are shown below:

Characteristics type	C.TYP	
Size of data input	C.DATI	
Size of data output	C.DATO	
Characteristics status	C.STAT	
Ethernet address		Characteristics Data
		6 bytes

Note

1. Each address entry is a three word quantity which contains the 48-bit Ethernet address.

4.5.2 Setting the Protocol Type

You must define the protocol type by setting the C.TYP entry to CC.PRO (101) before you can successfully transmit messages over the Ethernet channel. The individual characteristics are shown below:

Characteristics type	C.TYP	
Size of data input	C.DATI	
Size of data output	C.DATO	
Characteristics status	C.STAT	
Protocol type		Characteristics Data
		2 bytes

Note

1. The protocol type entry is a one word quantity which contains a user-defined 16-bit protocol value.

C.STAT Error Code

IE.BAD An attempt to transmit a message was rejected by the system. The auxiliary characteristics buffer did not include the Ethernet address and protocol type.

Completion Status Codes

IS.SUC The message was successfully transmitted to the remote node.
(1)

177733

IE.NLN No channel has been opened with the specified LUN.
(-37)

177761

IE.ABO The transmission was aborted and the channel was disconnected.
(-15) An unrecoverable error occurred in the hardware device. You must issue a QIO to close and then reopen the Ethernet channel (see Sections 4.7 and 4.3, respectively) before you can use that channel again.

177775

IE.DNR The hardware device was not ready. The channel was disconnected and has not been reinitialized.
(-3)

IO.XRC

Receive a Message on the Ethernet

4.6 IO.XRC – RECEIVE A MESSAGE ON THE ETHERNET

This macro call allows you to receive data from a sending task. When you receive a message on the Ethernet, you must find out the source Ethernet address for this message, the protocol type and the destination Ethernet address. To do this, you should use arguments *p3* and *p4* which specify the optional auxiliary characteristics buffer. You should then examine the C.DATO entry located in this buffer for the length of the message address and protocol information.

The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 4.4.

Format

```
QIO$ IO.XRC,lun,[efn],[status],[ast],
        <p1,p2,[p3,p4]>
```

Arguments

IO.XRC

is the function code for receiving a message.

lun

specifies the logical unit number used in the DLX open call.

efn

specifies an event flag number set when an IO.XRC call completes processing.

status

specifies the address of the 2-word status block containing completion status information on return from the call. This information is contained in the low-order byte of the first word.

ast

specifies the starting address of a user-written AST routine.

p1

specifies the address of the buffer which will receive the message.

p2

specifies the receive buffer length in bytes.

p3

specifies the address of the optional auxiliary characteristics buffer.

p4

specifies the length of the optional auxiliary characteristics buffer.

4.6.1 Optional Auxiliary Buffer for Receive Messages

The optional auxiliary buffer contains the following individual characteristics blocks as shown below:

Read the Ethernet Address

C.TYP = CC.ADR (100)

Characteristics type	C.TYP	
Size of data input	C.DATI	
Size of data output	C.DATO	
Characteristics status	C.STAT	
Ethernet address		Characteristics Data
		6 bytes

Read the Protocol Type

C.TYP = CC.PRO (101)

Characteristics type	C.TYP	
Size of data input	C.DATI	
Size of data output	C.DATO	
Characteristics status	C.STAT	
Protocol type		Characteristics Data
		2 bytes

Read Destination Ethernet Address

C.TYP = CC.DAD (102)

Characteristics type	C.TYP	
Size of data input	C.DATI	
Size of data output	C.DATO	
Characteristics status	C.STAT	
Destination Ethernet address		Characteristics Data
		6 bytes

C.STAT Completion Codes

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 received.)								
177666 IE.TMO (-74)	A timeout condition has occurred. No message was received within the specified timer interval when you opened or initialized the Ethernet channel.								
177733 IE.NLN (-37)	No line has been opened having the specified LUN.								
177761 IE.ABO (-15)	The receive function was aborted and the Ethernet channel was disconnected. An unrecoverable error occurred in the hardware device. You must issue a QIO to close and then reopen the Ethernet channel (see Sections 4.7 and 4.3, respectively) before you can use it again.								
177763 IE.DAO (-13)	The user buffer was not large enough to receive all of the data or the message was received before a receive QIO was issued by the user. The message is truncated and some of the data is lost during transmission. (The length of the user buffer is contained in the second word of the I/O status block.)								
177774 IE.VER (-4)	An error has occurred on the channel. The second word of the I/O status block contains the error code. Possible error codes and their meanings are: <table> <tr> <td>100362</td><td>Operation aborted</td></tr> <tr> <td>100363</td><td>Message received without receive pending</td></tr> <tr> <td>100364</td><td>Start received</td></tr> <tr> <td>100370</td><td>General error</td></tr> </table>	100362	Operation aborted	100363	Message received without receive pending	100364	Start received	100370	General error
100362	Operation aborted								
100363	Message received without receive pending								
100364	Start received								
100370	General error								
177775 IE.DNR (-3)	The hardware device was not ready. The channel was closed and has not been reinitialized.								

IO.XCL

Close the Ethernet Channel

4.7 IO.XCL – CLOSE THE ETHERNET CHANNEL

You should issue the IO.XCL call to close an open channel and stop the protocol.

Format

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

Arguments

IO.XCL

is the function code that closes the channel.

lun

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

efn

specifies an event flag number set when the IO.XCL call completes processing.

status

specifies the address of the 2-word status block containing completion status information in the low-order byte of the first word.

ast

specifies the starting address of a user-written AST routine.

Completion Status Codes

IS.SUC The channel has been successfully closed.
(1)

177733
IE.NLN No channel has been opened with the specified LUN.
(-37)

CHAPTER 5

REMOTE FILE ACCESS

Your Professional 350 computer is capable of storing large amounts of information on the hard disk and diskettes. You can access this information through RMS-11 which features data access capabilities.

PRO/DECnet supports extended RMS-11 capabilities which provide access to remote DECnet and RMS-11 programs. PRO/DECnet applications can perform these remote operations:

- ☐ Open an existing file
- ☐ Create a new file and define the internal structure of the file (the size and arrangement of records within the file)
- ☐ Read and write records within a file
- ☐ Close a file
- ☐ Delete a file

5.1 INTRODUCTION

Remote file access can only take place after a logical link is established between two cooperating network programs. When you open a file using RMS-11, a logical link is established between RMS-11 and a File Access Listener (FAL) program. FAL is responsible for performing all file operations on the remote system.

5.2 USING PRO/DECnet FOR REMOTE FILE ACCESS

When you use PRO/DECnet, your program can perform remote file access to different operating systems. RMS-11 offers an interface between any application program and the remote system's FAL. Since other Digital systems (including RSX and VMS systems) support RMS, your Professional 350 can share information with larger systems.

For most purposes, differences between local and remote access are transparent to the user. However, the following limitations apply:

- ☐ RMS-11 generally does not support remote functions (for example, to a VMS node) that are not supported locally.
- ☐ Certain RMS-11 functions (wildcard support; the PARSE, SEARCH, ENTER and REMOVE operations; and the use of default file specifications and file IDs) are not supported by the Data Access Protocol (DAP). As a result, they cannot be executed remotely.
- ☐ High level languages may not allow expression of the file specification required to establish contact with a remote node.

To operate upon a remote file, you must include the PRO/RMS-11 remote access code when you build your task. Your program must also include a node specification for the remote file. To include the remote access code, you must link your program with the RMS DAP modules by referencing the DAPRES resident library. You should include the entry DAPRES in the Professional Application Builder CLSTR option and use LB:[1,5]DAPRLX.ODL instead of RMSRLX.ODL. If you are using asynchronous RMS-11 operations, you must select special RMS-11 modules as indicated in DAPRLX.ODL. Refer to the *Tool Kit User's Guide* (AA-N617A-TK) for additional information.

Here is a sample build file:

```
TASK=TASK
/
CLSTR=RMSRES,DAPRES:RO
//
```

NOTE

RMS-11 uses the file access block (FAB) logical channel number as the link ID for remote access. Users performing remote access external to RMS-11 should be careful not to use the same link IDs.

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.

If RMS returns an error code of ER\$FAL, refer to Appendix G for an interpretation of the specific STV error code.

5.3 FORMATTING REMOTE NODE SPECIFICATIONS

You must include a remote node specification as the first element of the file name string or the default name string specified for an OPEN, CREATE or ERASE operation. Your file name and default name strings must conform to the rules for Digital Command Language (DCL) file specification. In addition, the file specification which results from merging the file name and default name strings must conform to the conventions of the target node.

A full remote file specification normally has this format:

node::device:[directory]name.type;version

Elements beyond the node name must conform to DCL syntax and the conventions of the target node. If the file name string does not provide all six elements, RMS-11 obtains missing elements from the default name string. After the two strings have merged, any elements still unspecified are defaulted according to the conventions of the target system.

An alternative remote file specification format is:

node::"quotedstring"

where *quotedstring* is any file specification that conforms to target system conventions. For example, this format provides a means of passing certain RSTS/E logical names (\$,% , etc.) that do not conform to DCL conventions.

If the quoted string contains any quotation mark (") characters, you must insert an additional quotation mark before each one. These extra quotation marks will be stripped away when the string is passed to the target system. Any elements not present in the quoted string will be defaulted according to the target system conventions.

RMS-11 treats specifications of this format as complete indivisible specifications. If one occurs in the file name string, no elements from the default name string will be used; if one occurs in the default name string, it will be ignored unless the file name string is empty.

The node element takes this form:

node"user password"

where *node* is the destination node name and *user password* is an optional access control string containing log-in information (user ID and password, separated by a space character) that meets target system log-in conventions.

If log-in information is provided, the device and directory default and the access privileges of the remote account are used by FAL. Otherwise, the device and directory defaults and access privileges of the default DECnet account on the target system are used by FAL. If the alias contains the required access control information, you do not have to specify the node element.

5.4 REMOTE ACCESS ENVIRONMENTS

The PRO/RMS-11 DAP routines only support access to RMS-based FALs. These FALs are currently available on VAX/VMS, RSTS/E, and RSX-11M/M-PLUS systems. The version of DAP supported by the remote FAL must be at least Version 5.6 or later. This particular FAL version also requires at least DECnet/E Version 2.0, DECnet for RSX-11M Version 3.1, DECnet for RSX-11M/M-PLUS Version 1.1, or DECnet VAX Version 2.0.

5.5 REMOTE ACCESS POOL CONSIDERATIONS

Remote block access, unlike local block access, requires an internal I/O buffer for both record operations and the initial OPEN or CREATE operation. This I/O buffer is reserved for the file while it is open, and must be 548 bytes in size.

Similarly, for a sequential file with a maximum record size (or actual largest record) greater than 476 bytes, an internal I/O buffer must exceed this size by 36 bytes while the sequential file is open. For record access relative and indexed files, an internal I/O buffer equal to the bucket size is required while the file is open. Other pool requirements are equal to or less than those for local access.

See the *PRO/RMS-11 Macro Programmer's Guide* for details on file structure.

APPENDIX A

BASIC DECnet CONCEPTS

A.1 TASK-TO-TASK COMMUNICATION

DECnet enables two programs within a network to perform task-to-task communication – that is, to exchange data over a logical link. DECnet calls are written into cooperating task programs on different nodes. The cooperating tasks can be written in MACRO-11, FORTRAN, COBOL, BASIC, or PASCAL.

PRO/DECnet tasks support the same task-to-task I/O calls used by DECnet-RSX tasks. As a result, an RSX network program could easily be converted to run on your Professional 350.

DECnet communication calls activate routines which request a local DECnet node to perform certain network functions. Every DECnet task-to-task program can issue calls to perform these functions:

- ☐ Establish an active network task
- ☐ Build a connect block
- ☐ Establish a logical link
- ☐ Retrieve data from a task's network data queue
- ☐ Send and receive data messages
- ☐ Check completion status information
- ☐ Terminate activity on a logical link
- ☐ Close a network connection

A.2 ESTABLISHING AN ACTIVE NETWORK TASK

Before a task can exchange data over the network using DECnet communication calls, it must become active by first issuing a DECnet open call (OPN\$ or OPNNT). This step provides the task with a network data queue and connects the task to the DECnet communications facilities. The DECnet software passes any connect requests and interrupt, disconnect, and abort messages from other tasks to the task's network data queue.

It is possible for you to limit the number of connect requests that the task will accept and pass along to its network data queue. The DECnet software will automatically reject any additional connect requests. The number of connect requests can range from 0 to 255, with 0 set as the default value. A value of 0 implies an unlimited number of incoming connect requests which can be placed on the task's network data queue. In this situation, the task is responsible for accepting or rejecting the incoming connect requests.

A.3 BUILDING A CONNECT BLOCK

Before a task can issue a connect request, it must first build a connect block. You should reserve a 72 byte area of memory for the connect block information. This area must begin on a word or even byte boundary. The connect block contains specific parameters which identify the source and target tasks. A connect block contains a destination descriptor, a source descriptor, access control information for remote file access, and optional user data. MACRO-11 tasks use the DECnet macro call CONB\$\$ to build connect blocks. The high level languages use a pair of DECnet subroutine calls: the build access control area call (BACC) with either the build format 0 descriptor (BFMT0) or the build format 1 descriptor (BFMT1) call.

A.3.1 Destination Descriptor

Network object is another name for a network program which is only responsible for accepting incoming connect requests. This program does not issue any connect requests. A network program or object has a special identifier for use in DECnet connect requests. This identifier can be an object name or an object type code. The destination descriptor identifies the target task by its object name or by its object type code.

Named objects are user-written tasks which are referenced by a name during a connect request. The object type code for such tasks is 0.

- ☐ High level language tasks specify 0 for a named object and the task name as the descriptor in the BFMT1 call.
- ☐ MACRO-11 tasks specify 0 for a named object, 1 as the descriptor format type and the task name in the CONB\$ call.

Numbered objects are installed tasks which are referenced by an object type code. The object type numbers range from 1 to 255. Numbers 1 to 127 are reserved for DECnet-specific tasks. Numbers 128 to 255 are reserved for user-written tasks. (See Appendix C for more information on object type codes.)

- ☐ High level language tasks specify the object type codes in the BFMT0 call.
- ☐ MACRO-11 tasks specify the object type codes along with 0 as the descriptor format type in the CONB\$ call.

A.3.2 Source Descriptor

The source descriptor contains the source node name and either the source task's object name or its object number type. The contents of the source descriptor are supplied by the DECnet software on the source node.

A.3.3 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. For some target systems, the access control information is verified before a connect request is passed to the target task.

Access control information consists of a user ID and password, and sometimes an account number on the target node. Other information may be required such as directory names and device names for remote file access. You should always review the appropriate documentation for each target system's requirements.

A.3.4 Optional Data Message

When the source task issues a connect request, you have the option of including a data message up to 16 bytes long in the connect block. If the connect call contains the optional data message, the source node's DECnet software will append the message to the connect block. An optional data message can also be sent with a DECnet disconnect or a reject request.

A.4 ASSIGNING LOGICAL UNIT NUMBERS

A logical link identifier is used by the source task whenever it accesses the network, and establishes a temporary logical link with a target task. It is also used for all task-to-task communication between the two tasks. This identifier can be specified in a DECnet open call (OPN\$ or OPNNT), a connect call (CON\$ or CONNT), and an accept call (ACC\$ or ACCNT), or during task build

time. For PRO/DECnet applications, the logical link identifier is known as a logical unit number (LUN). When you assign LUNs for a COBOL DECnet call, do not use LUN 1. It is a reserved number. Likewise, for a PASCAL DECnet call, you should assign LUNs 25 to 40 for DECnet-specific operations and reserve 50 to n for file operations.

Assigned LUNs can be found in these DECnet calls:

- ☐ OPN\$ or OPNNT (Access the network)
- ☐ CON\$ or CONNT (Request a logical link connection)
- ☐ ACC\$ or ACCNT (Accept a logical link request)
- ☐ REJ\$ or REJNT (Reject a logical link request)
- ☐ GND\$ or GNDNT (Get network data)
- ☐ GLN\$ or GLNNT (Get local node information)
- ☐ SND\$ or SNDNT (Send a message over a logical link)
- ☐ REC\$ or RECNT (Receive a message over a logical link)
- ☐ SPA\$ (Specify a user-written AST routine)
- ☐ ABT\$ or ABTNT (Abort a logical link)
- ☐ DSC\$ or DSCNT (Disconnect a logical link)
- ☐ CLS\$ or CLSNT (End a task's network operation)

A.5 ESTABLISHING A LOGICAL LINK

The creation of a logical link is a cooperative venture. Two tasks must agree to communicate before you can have an established logical link. The task requesting a logical link is called the source task. The task that accepts or rejects the request is called the target task or network object. The process begins when a source task issues a DECnet connect call (CON\$ or CONNT). This call includes the connect block which specifies the target node name and target task, and the assigned LUN.

A task can attempt to establish as many logical links as required for a specific application. The number of logical links is limited only by the amount of available system resources. However, the logical link count can never exceed a maximum of 255.

When a target system receives a connect request, it checks to see if the target task is installed on the network. If it is, the target task is automatically loaded and activated by the DECnet software. The incoming connect block is placed on the target task's network data queue.

The maximum size of a connect block is normally 98 bytes. If an optional data message is included with a connect request, the connect block's maximum size cannot exceed 114 bytes. (An optional data message has a maximum size of 16 bytes.)

The target system examines the access control information in the connect block and checks the validity of the object type code. If everything checks out, the connect block is passed along to the target task.

Figure A-1 illustrates the flowchart process for establishing a logical link.

A.6 GETTING DATA FROM THE NETWORK DATA QUEUE

Once a task is connected to the network using the DECnet open call (OPNNT or OPN\$), it becomes an active network task with its own network data queue. A task should begin to monitor its network data queue immediately following a successful open call. When a target task issues a get network data call (GNDNT or GND\$), the incoming connect block is made available to it.

The target task places the connect request in its buffer. It then evaluates the connect request and decides to either accept or reject it. If the target task's buffer cannot hold an optional data message, the get network data call completes with a data overflow condition.

The DECnet software also places interrupts, user disconnects, user aborts, and network aborts on the task's network data queue. The same DECnet call (GND\$ or GNDNT) is used to retrieve and examine these messages.

The get network data call usually returns the oldest message on a first-in, first-out basis. However, there are other options available for removing these messages. (See Sections 2.10.11 and 3.9.9.)

A.7 ACCEPTING OR REJECTING A LOGICAL LINK CONNECTION REQUEST

When a get network data call returns a connect request, the target task either accepts or rejects the connect request before issuing another call.

The target task follows one of these procedures for accepting or rejecting the connect request:

1. The target task accepts the connect request by issuing a DECnet accept call (ACC\$ or ACCNT). This call includes the assigned LUN for the logical link and the location of the buffer containing the connect block information.
2. If the target chooses to reject the connect request, it must issue a DECnet reject call (REJ\$ or REJNT). The buffer containing the connect block is also specified in this call.

NOTE

You can specify the location and length of an optional data message for the source task in both the accept and reject calls. This message can be up to 16 bytes long. If the source task's buffer is not large enough to handle the message, the operation will result in a data overflow condition.

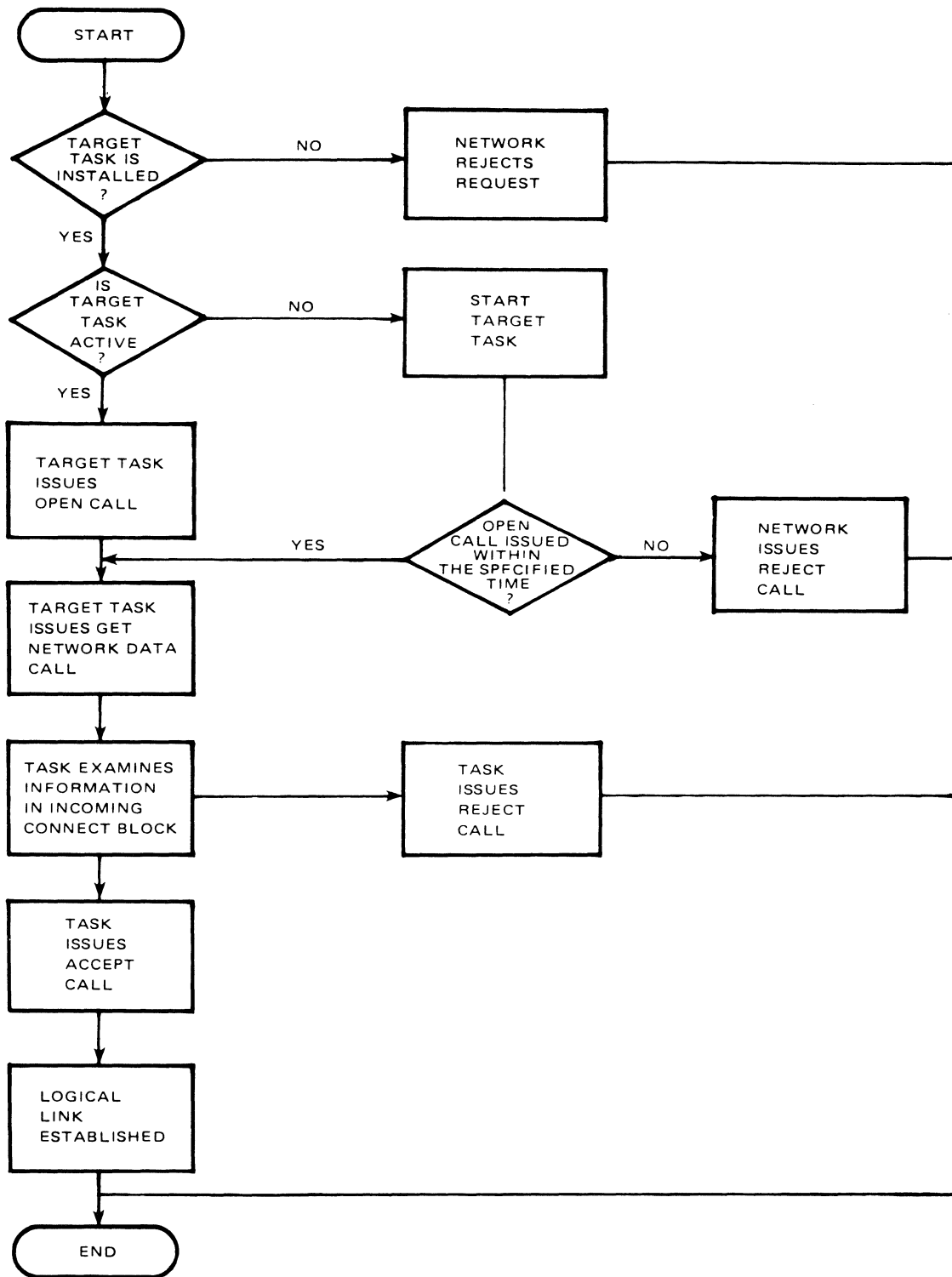


Figure A-1
Establishing a Logical Link

3. The accepted or rejected connect request is passed over the temporary logical link via DECnet to the source task. If the source task accepts the connection, the temporary logical link becomes permanent. If it rejects the connection, the temporary logical link is disconnected.

Once a logical link has been established between two tasks, both tasks can send and receive data and unsolicited, high priority messages over it. DECnet distinguishes between data and high priority messages. It delivers the latter to a task's network data queue. It delivers data messages directly to a buffer provided by the receiving task.

A.8 TRANSMITTING DATA MESSAGES OVER A LOGICAL LINK

When messages are transmitted over a logical link, the LUN assigned during an accept or connect sequence is referenced with each send and receive call. The DECnet software allows tasks to send and receive data at the same time. This capability is known as logical full duplex transmission.

Before issuing its first send or receive call, a task can issue a get local node data call (GLN\$ or GLNDT). This call returns the arbitrated segment buffer size which allows DECnet to transmit messages in the most efficient way.

A.8.1 Sending Data Messages

A task must issue a DECnet send call (SND\$ or SNDNT) in order to send a data message. This call specifies the assigned LUN used for transmitting the message and the buffer containing the outgoing message.

At any time, a task can have several send calls awaiting completion. A send call completes when DECnet on the receiving node informs DECnet on the sending node that it received the data. For most cases, this completion notice means that DECnet received the message and not the target task.

A.8.2 Receiving Data Messages

A task must issue a DECnet receive call (REC\$ or RECNT) before it can receive any data messages. The receive call specifies the logical link which will receive the message, and allocates the required buffer space to store the incoming message.

A task can issue multiple receive calls before any messages are sent by the other task. This procedure helps insure continuous data flow between the two tasks. A receive call completes when the DECnet software moves the message into the specified buffer. You can verify message reception by periodically checking the contents of word 0 of the I/O status block (see Sections 2.5 and 3.4); by using WAITNT (see Section 2.10.16) or via synchronous or asynchronous completions (see Section 3.3).

A.9 SENDING INTERRUPT MESSAGES

Interrupt messages are unsolicited, high priority messages sent between tasks over a logical link. An interrupt message usually informs the receiving task of an unusual or abnormal event in the sending task. Special interrupt messages are issued directly by DECnet. For example, if a physical connection breaks between two nodes, and no alternate physical path is available, an abort message is sent to all tasks having logical links over that physical connection.

Normal data messages are always received in the order sent by the sending task. However, interrupt messages may be delivered to a receiving task's network data queue ahead of normal data messages.

Several logical links may exist between a source task and one or more target tasks. It is also possible for a receiving task to send many interrupt messages over different logical links. A target task can also have many interrupt messages contained in its network data queue. However, it cannot have more than one interrupt message for any single logical link. For this reason, an interrupt message must be removed from the queue before a second interrupt message is sent over the same logical link.

DECnet flow control does not apply to interrupt messages. These messages are delivered to a task's buffer even if it has no outstanding receive calls.

A task must issue a DECnet interrupt call (XMI\$ or XMINT) before it can send an interrupt message. This call specifies the logical link used for sending the message and the buffer containing the interrupt message. The size of the interrupt message cannot exceed 16 bytes.

An interrupt call completes when DECnet on the receiving node informs DECnet on the sending node that it received the message. The completion notice means only that DECnet, not the task, has received the message.

The interrupt message is then placed on the receiving task's network data queue. A get network data call (GND\$ or GNDNT) is issued to remove the message from the queue and place it in the task's buffer. At this point, a new interrupt message can be issued to the receiving task on the same logical link.

A.10 USING THE I/O STATUS BLOCK

Most calls pass an I/O status block as an argument to the DECnet call. Even though *status* is considered to be an optional argument, it is the only way for checking the status of a returned call. Moreover, each active macro and subroutine should have its own I/O status block. Using the same I/O status block for several concurrent calls can produce unpredictable results during task execution.

The I/O status block consists of two words. The low-order byte of the first word (word 0) contains the error completion code. This byte can have three different values:

- ☐ A null value (0) indicates an incomplete macro or subroutine call.
- ☐ A positive value indicates a successfully completed macro or subroutine call.
- ☐ A negative value shows that the macro or subroutine call did not produce the desired results.

The second word of the I/O status block contains more completion information. For example, the number of bytes received in a successful data transmission is contained in the second word.

A.11 TERMINATING ACTIVITY ON A LOGICAL LINK

A task can issue either a disconnect or an abort call for terminating activity on a logical link. An optional message having up to 16 bytes of data can be sent along with either message. All disconnect and abort messages are placed on the other task's network data queue. This task must issue a get network data call (GND\$ or GNDNT) in order to retrieve them.

A.11.1 Disconnecting a Logical Link

A task must issue a disconnect call (DSC\$ or DSCNT) before it can dissolve a logical link. A disconnect call causes an orderly termination of transmissions over the logical link. All pending transmits from the issuing task are completed before DECnet actually disconnects the logical link. The issuing task continues to receive incoming messages for a predetermined period of time. Once this time period elapses, any remaining receive calls are aborted by the system. DECnet then dissolves the logical link and releases the assigned LUN for use in a subsequent connect or accept call.

A.11.2 Aborting a Logical Link

A task must issue a DECnet abort call (ABT\$ or ABTNT) before it can abort a logical link. This call immediately aborts all pending transmits and receives, and disconnects the logical link. The assigned LUN is also released and made available for use in a subsequent connect or accept call.

A.12 CLOSING THE NETWORK CONNECTION

The DECnet close call (CLS\$ or CLSNT) is issued when a task no longer requires network services. This call informs DECnet to purge the task's network data queue. Any active LUNs are deactivated and ready for use in a subsequent DECnet open call.

If the task's network data queue contains data when the close call is issued, one of these events can occur:

1. The task becomes reactivated whenever there are pending connect requests in the queue. It can receive the new requests by subsequently issuing a DECnet open call (OPN\$ or OPNNT).
2. All interrupt, disconnect, and abort messages will be ignored by the system.

A.13 DECnet TASK-TO-TASK COMMUNICATION CALLS

DECnet task-to-task communication calls are used with high level language tasks as well as MACRO-11 tasks. The format of a DECnet call is determined by the programming language.

Certain DECnet calls are only used with MACRO-11 tasks. As a result, these calls have only one version. Table A-1 lists the calls and describes their functions.

NOTE

All MACRO-11 calls described in this table and in the following sections include the dollar symbol (\$) as part of the call's syntax. High level language calls do not use the dollar symbol (\$) as part of their syntax.

Table A-1
DECnet Communication Calls Summary

ABT\$ or ABTNT

An abort is a high priority message transmitted over the logical link. It is delivered to the target task's network data queue. It completes any pending send or receive calls with an immediate abort error and dissolves the logical link.

ACCS\$ or ACCNT

A target task issues this call in order to accept a logical link connect request from another task. When this call is issued, a notification is sent to the source task.

BACC

The BACC call specifies access control information for a connect block. In a succeeding CONNT call, the target task determines if the access control string contains all the necessary specifications for remote file access.

BFMT0

This call specifies the descriptor block for a target task installed as a numbered object. This block identifies the task by its object type and not by its name.

BFMT1

The BFMT1 call specifies the descriptor block for a target task installed as a named object. The block identifies the task by its name and not by its function.

CLS\$ or CLSNT

A close request is used to terminate an issuing task's connection with the network. This call completes any pending send or receive calls with an immediate abort and dissolves all logical links.

CON\$ or CONNT

A task issues this call to request a connection to a target task. The initial connection is set up via a temporary logical link. If the target task is inactive, it is automatically started by the network. The connect request is then delivered to the target task's network data queue. The target task issues a get network data call (GND\$ or GNDNT) and the connect block is moved from the queue to a buffer.

If the target task accepts the connect request, the temporary logical link becomes permanent.

CONB\$\$

This call builds a data area which is used as a connect block in a succeeding CON\$ macro call.

(continued on next page)

A-12 BASIC DECnet CONCEPTS

Table A-1 (cont)
DECnet Communication Calls Summary

DSC\$ or DSCNT

This call causes an orderly disconnect of a logical link. The disconnect call can be used by either the source or target task. The logical link disconnects after all pending messages are delivered to the cooperating task.

GLN\$ or GLNNT

This call allows your programs to retrieve three data items: the local node name and address, and the default segment buffer size used by the network.

GND\$ or GNDNT

This call is used by both the source and target tasks to retrieve any messages on their network data queues. The target task will usually make this call after issuing an open call in order to retrieve the source task's connect call.

OPN\$ or OPNNT

This call informs the network that the issuing task requires network services, and causes the network to create the task's network data queue. Both the source and target tasks must issue this call prior to making any other DECnet calls.

REC\$ or RECNT

This call requests reception of a message sent by the cooperating task. It identifies a task buffer used for storing incoming data messages.

REJ\$ or REJNT

This call rejects a connect request from the source task. The rejection message is returned to the I/O status block of the source task's CON\$ or CONNT call.

SND\$ or SNDNT

This call sends a data buffer to a receiving task over an established logical link. Once the data is received by the other system, a completion status message is returned to the I/O status block of the send call.

SPA\$

This call provides a task with a way of monitoring and reacting to network events. It specifies a user-written asynchronous system trap (AST) routine which will execute whenever data arrives on a task's network data queue.

WAITNT

This call suspends the execution of a task until a previously issued call has completed processing. It signals the completion of another call in no-wait form.

XMI\$ or XMINT

This call allows you to transmit an interrupt message to a receiving task. The interrupt message is sent over the logical link to the receiving task's network data queue.

APPENDIX B

DISCONNECT OR REJECT REASON CODES

The following list contains the error codes available at the logical link user interface. These codes can be returned in the third byte of the status block after one of these events occurs:

- ☐ A connect request was rejected by the network (IE.NRJ). (See Section 3.10.5.)
- ☐ A connected logical link was aborted by the network (NT.ABO). (See Section 3.10.2.)

The symbols in column one are defined in the macro NSSYM\$. NSSYM\$ is located in NETLIB.MLB and is provided on the Tool Kit. The events in column five indicate the error results. "C" refers to a connect request and "A" refers to a network abort.

B-2 DISCONNECT OR REJECT REASON CODES

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
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 specified object number or name 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 was aborted by an operator or a program using network management.	A

(continued on next page)

DISCONNECT OR REJECT REASON CODES B-3

Symbol Name	Decimal Value	Octal Value	Standard Message/Explanation	Event
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 was too long or contained illegal characters.	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
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

APPENDIX C

OBJECT TYPES

The following object type code values have been defined by Digital. They are 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 named object (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		Process Type
Octal	Decimal	
000	000	General task or user process connected to by task name
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

(continued on next page)

C-2 OBJECT TYPES

Object Type		Process Type (cont)
Octal	Decimal	
012	010	IBM 3790-SDLC Gateway
013	011	TPS application
014	012	RT-11 DIBOL application
015	013	TOPS-20 terminal handler
016	014	TOPS-20 remote spooler
017	015	RSX-11M Remote Task Control utility (TCL) - Version 2
020	016	Network Talk 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

APPENDIX D

MACRO-11 CONNECT BLOCK OFFSETS AND CODE DEFINITIONS

The following MACRO-11 connect block offsets are 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$
```

(continued on next page)

D-2 MACRO-11 CONNECT BLOCK OFFSETS AND CODE DEFINITIONS

++
; REQUEST DESCRIPTOR BLOCK OFFSET DEFINITIONS FOR CONNECTS.
;
;

N,RND	{ 000
	{ 004
N,ROT N,RFM	006

FORMAT 0

(UNUSED)	{ 010
	{ 030

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

(continued on next page)

```

        .MACRO CRBDF$,L,B,LST
        .if 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

        .=-18.         ; 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

        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
        .if nb LST      .Nlist
        .if
        .MACRO CRBDF$,X,Y,Z
        .ENDM CRBDF$
        .ENDC

```

(continued on next page)

D-4 MACRO-11 CONNECT BLOCK OFFSETS AND CODE DEFINITIONS

```

; CONNECT BLOCK OFFSET DEFINITIONS FOR RECEIVED COUNT
; REQUESTS.

```

N.CTL		000
N.SEGZ		002
N.DOT	N.DFM	005/004

FORMAT 0

(UNUSED)	{ 006 026
----------	--------------

FORMAT 1

N.DDEC	006	
N.DDE	{ 010 026	
N.SND	{ 030 034	
N.SOT	N.SFM	037/036

(continued on next page)

;	FORMAT 0	
;		
;	(UNUSED)	{ 040
;		{ 060
;	FORMAT 1	
;	N.SDEC	040
;	N.SDE	{ 042
;		{ 060
;	No verification performed	
;	N.CIDC	062
;	N.CID	{ 064
;		{ 102
;	N.CPSC	104
;	N.CPS	{ 106
;		{ 114
;	N.CACC	116
;	N.CAC	{ 120
;		{ 136
;	N.CDAC	140
;	Verification performed	
;	N.CDEV	062
;		
;	N.CUNI	064
;	N.CUIC	066
;		
;		140
;	N.CDA	142
;		
;		
;		

(continued on next page)

D-6 MACRO-11 CONNECT BLOCK OFFSETS AND CODE DEFINITIONS

```

,PSECT
.ENDM CRBDF$
,MACRO CNBDF$,L,B,LST
,if 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

,=,
,BLKB 18, ; Format 0
; Unused

,=-18,
N,DDEC:'L' ,BLKW 1 ; Destination Process byte count
N,DDE:'L' ,BLKB 16, ; Destination Process

,=-18,
N,DGP:'L' ,BLKW 1 ; Format 2
; 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

,=,
,BLKB 18, ; Format 0
; Unused

,=-18,
N,SDEC:'L' ,BLKW 1 ; Format 1
; Source Process name byte count
N,SDE:'L' ,BLKB 16, ; Source Process name

,=-18,
N,SGP:'L' ,BLKW 1 ; Format 2
; Source group
N,SUS:'L' ,BLKW 1 ; Source user
N,SNMC:'L' ,BLKW 1 ; Source name byte count
N,SNM:'L' ,BLKB 12, ; Source name

$$$=,
N,CIDC:'L' ,BLKW 1 ; Source task ID byte count
N,CID:'L' ,BLKB 16, ; Source task ID byte count
N,CPSC:'L' ,BLKW 1 ; Password byte count
N,CPS:'L' ,BLKB 8, ; Password
N,CACC:'L' ,BLKW 1 ; Accounting information byte count
N,CAC:'L' ,BLKB 16, ; Accounting information
N,CDAC:'L' ,BLKW 1 ; Optional data byte count
N,CDA:'L' ; Optional data
;
N,CBL='B',-N,CTL ; Length of CNB (without any data)

```

(continued on next page)

```

,=$$$$
N,CDEV:'L'      ,BLKW  1      ; Default device name (from account file)
N,CUNI:'L'      ,BLKB  1      ; Default device unit number
                ,EVEN
N,CUIC:'L'      ,BLKW  1      ; Log in UIC (from account file)

                ,PSECT
                ,if nb  LST      ,Nlist
                ,iff
                ,MACRO  CNBDF$,X,Y,Z
                ,ENDM    CNBDF$
                ,endc
                ,ENDM    CNBDF$

```


APPENDIX E

ERROR/COMPLETION CODES FOR HIGH LEVEL LANGUAGES

The following error/completion codes are returned in the first word of a 2-word I/O status block.

- 1 The request was successful.
- 2 The request was successful, but some optional data was lost.
- 1 The required system resources are not available.
- 2 A request was issued for a LUN having no established logical link.
- 3 The link was disconnected with an outstanding request.
- 4 The received data was truncated due to an insufficient receive buffer length.
- 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 rejected an attempted connect.
- 8 The specified logical unit number is already in use on another logical link. The user was unable to connect to the link using the same LUN.
- 9 The issuing task is not a network task. OPNNT did not execute successfully.
- 10 The network has already been opened using an OPNNT call.
- 11 Transmission of an interrupt message was attempted before the last one finished.
- 12 A connect/reject was issued by the user task for an attempted connection.
- 13 Either a buffer is outside the user task address space or is not word aligned.
- 14 The user is attempting to issue a GNDNT[W] when one is already pending.
- 40 A directive error has occurred. The second word of the I/O status block contains the actual directive error code.

APPENDIX F

MACRO-11 ERROR/COMPLETION CODES

This appendix contains only those MACRO-11 error/completion codes described in this manual.

<i>Mnemonic</i>	<i>Decimal Value</i>	<i>Octal Value</i>	<i>Meaning</i>
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	Either an invalid buffer parameter or the data length exceeds 16-bytes.
IE.SPC	-6	372	Invalid buffer parameters: buffer may not be word aligned; buffer may be outside user task 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 Appendix B.)
IE.PRI	-16	360	The network is not accessed on this LUN.
IE.RSU	-17	357	The required system resources are not available.
IE.ALN	-34	336	The specified LUN is already in use.

(continued on next page)

F-2 MACRO-11 ERROR/COMPLETION CODES

Mnemonic	Decimal Value	Octal Value	Meaning
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 Appendix B.)
IE.NDA	-78	262	There is no data to return.
IE.NNT	-94	242	The issuing task is not a network task. OPN\$ did not execute successfully.

APPENDIX G

SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

G.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 G-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.

G-2 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-1
First Word I/O Status Block Error Codes

<i>Error Code</i>	<i>Description</i>
177777 (-1)	<p>CHANNEL ALREADY ACTIVE</p> <p>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.</p> <p>The second word of the I/O status block is not applicable.</p>
177776 (-2)	<p>CHANNEL NOT ACTIVE</p> <p>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.</p> <p>The second word of the I/O status block is not applicable.</p>
177775 (-3)	<p>DATA ACCESS PROTOCOL ERROR</p> <p>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.</p> <p>The second word of the I/O status block contains the file access error code. Look up this code in Table G-3.</p>
177774 (-4)	<p>NSP ERROR (See Table G-2)</p> <p>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.</p> <p>The low-order byte of the second word of the I/O status block contains one of the NSP error codes listed in Table G-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 B).</p>
177773 (-5)	<p>INVALID ATTRIBUTES</p> <p>An invalid character has been found in the attributes array (<i>ichar</i>) of an open command.</p>
177772 (-6)	<p>DATA OVERRUN</p> <p>A message or block of messages was received that did not fit into the user-specified buffer.</p> <p>The second word of the I/O status block contains the total number of bytes read.</p>

(continued on next page)

Table G-1 (cont)
First Word I/O Status Block Error Codes

177771 (-7)	TASKS OUT OF SYNC	<p>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.</p> <p>The second word of the I/O status block is not applicable.</p>
177770 (-8)	INVALID DAP CHANNEL (LUN)	<p>DAP channel numbers must fall in the range of 1 to 255. A 0 channel or a channel value greater than 255 is invalid.</p> <p>The second word of the I/O status block is not applicable.</p>
177767 (-9)	BUFFER ALLOCATION ERROR FOR DAP CHANNELS	<p>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.</p> <p>The second word of the I/O status block is not applicable.</p>
177766 (-10)	DIRECTIVE ERROR	<p>Directive error from the executive.</p> <p>The second word of the I/O status block contains the DSW value.</p>
177765 (-11)	ILLEGAL REQUEST	<p>An illegal request was made (for example, an attempt to read from a file that was open for write).</p> <p>The second word of the I/O status block is not applicable.</p>

Table G-2 contains the NSP error codes that pertain to the NSP ERROR in Table G-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.

G-4 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-2
NSP Error Codes

<i>Error Code</i>	<i>Description</i>
-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 B).
-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.

G.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 G-3 in Section G.2.1)
- ☐ Miccode (bits 0-11): Contains the specified error reason code (see Tables G-4, G-5, and G-6, depending on error type, as described in Section G.2.2)

G.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 G-3 are found in the table referenced in the last column of Table G-3.

Table G-3
DAP Maccode Field Values

<i>Field Value</i>	<i>Error Type</i>	<i>Meaning</i>	<i>Miccode Table</i>
0	Pending	The operation is in progress.	G-5
1	Successful	Returns information that indicates success.	G-5
2	Unsup-ported	This implementation of DAP does not support specified request.	G-4
3	Reserved		
4	File open	Errors that occur before a file is successfully opened.	G-5
5	Transfer error	Errors that occur after a file is opened and before it is closed.	G-5
6	Transfer warning	For operations on open files, indicates that the operation completed, but not with complete success.	G-5
7	Access termination	Errors associated with terminating access to a file.	G-5
10	Format	Error in parsing a message. Format is not correct.	G-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).	G-4
12	Sync	DAP message received out of synchronization.	G-6
13-15	Reserved		
16-17	User-defined status maccodes		

G.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 G.2.1). Miccode field values are defined in three different tables, each table associated with certain maccode values, as outlined below:

- ☐ Table G-4: For use with maccode values 2, 10, 11
- ☐ Table G-5: For use with maccode values 0, 1, 4, 5, 6, 7
- ☐ Table G-6: For use with maccode value 12

Table G-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 G-4
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
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	Bit count 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 G-4 (cont)
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
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)
	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)

(continued on next page)

G-8 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-4 (cont)
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
Access message errors		
03	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Access function field (ACCFUNC)
	21	Access options field (ACCOPT)
	22	File specification field (FILESPEC)
	23	File access field (FAC)
	24	File-sharing field (SHR)
	25	Display attributes request field (DISPLAY)
	26	File password field (PASSWORD)
Control message errors		
04	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Control function field (CTLFUNC)
	21	Control menu field (CTLMENU)
	22	Record access field (RAC)
	23	Key field (KEY)
	24	Key of reference field (KRF)
	25	Record options field (ROP)
	26	Hash code field (HSH); reserved for future use
	27	Display attributes request field (DISPLAY)
	30	Block count (BLKCNT)

(continued on next page)

Table G-4 (cont)
 DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
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)
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)

(continued on next page)

G-10 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-4 (cont)
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
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)
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)

(continued on next page)

Table G-4 (cont)
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
Key definition message errors		
12	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Key definition menu field (KEYMENU)
	21	Key option flags field (FLG)
	22	Data bucket fill quantity field (DFL)
	23	Index bucket fill quantity field (IFL)
	24	Key segment repeat count field (SEGCNT)
	25	Key segment position field (POS)
	26	Key segment size field (SIZ)
	27	Key of reference field (REF)
	30	Key name field (KNM)
	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)

(continued on next page)

G-12 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-4 (cont)

DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
Allocation message errors		
13	00	Unknown field
	10	DAP message flags field (FLAGS)
	11	Data stream identification field (STREAMID)
	12	Length field (LENGTH)
	13	Length extension field (LEN256)
	14	Bit count field (BITCNT)
	20	Allocation menu field (ALLMENU)
	21	Relative volume number field (VOL)
	22	Alignment options field (ALN)
	23	Allocation options field (AOP)
	24	Starting location field (LOC)
	25	Related file identification field (RFI)
	26	Allocation quantity field (ALQ)
	27	Area identification field (AID)
	30	Bucket size field (BKZ)
	31	Default extension quantity field (DEQ)
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 G-4 (cont)
DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
Data 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)

G-14 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-4 (cont)

DAP Miccode Values for Use with Maccode Values of 2, 10, 11

<i>Type Number (bits 6-11)</i>	<i>Field Number (bits 0-5)</i>	<i>Field Description</i>
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 G-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 G-5
DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
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

(continued on next page)

G-16 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-5 (cont)
DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
30	ER\$CRE	F11-ACP could not create file (STV = system error code)
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
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

(continued on next page)

Table G-5 (cont)

DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
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\$IIRC	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	F11-ACP 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
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)

(continued on next page)

G-18 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-5 (cont)

DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
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
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

(continued on next page)

Table G-5 (cont)
DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
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
172	CDA	Cannot deliver AST
173	CHN	Channel assignment failure (STV = system error code)
174	CNTRLO	Terminal output ignored due to CTRL/O
175	CNTRLY	Terminal input aborted due to CTRL/Y
176	DNA	Default file name string address error
177	DVI	Invalid device ID field
200	ESA	Expanded string address error
201	FNA	File name string address error
202	FSZ	FSZ field invalid
203	IAL	Invalid argument list
204	KFF	Known file found
205	LNE	Logical name error
206	NOD	Node name error
207	NORMAL	Operation successful
210	OK__DUP	Inserted record had duplicate key
211	OK__IDX	Index update error occurred; record inserted
212	OK__RLK	Record locked, but read anyway
213	OK__RRV	Record inserted in primary key is okay; may not be accessible by secondary keys or RFA
214	CREATE	File was created, but not opened
215	PBF	Bad prompt buffer address
216	PNDING	Asynchronous operation pending completion
217	QUO	Quoted string error
220	RHB	Record header buffer invalid
221	RLF	Invalid related file

(continued on next page)

G-20 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES


Table G-5 (cont)
DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
222	RSS	Invalid resultant string size
223	RST	Invalid resultant string address
224	SQO	Operation not sequential
225	SUC	Operation successful
226	SPRSED	Created file superseded existing version
227	SYN	File name syntax error
230	TMO	Timeout period expired
231	ER\$BLK	FB\$BLK record attribute not supported
232	ER\$BSZ	Bad byte size
233	ER\$CDR	Cannot disconnect RAB
234	ER\$CGJ	Cannot get JFN for file
235	ER\$COF	Cannot open file
236	ER\$JFN	Bad JFN value
237	ER\$PEF	Cannot position to end of file
240	ER\$TRU	Cannot truncate file
241	ER\$UDF	File currently in an undefined state; access is denied
242	ER\$XCL	File must be opened for exclusive access
243		Directory full
244	IE.HWR	Handler not in system
245	IE.FHE	Fatal hardware error
246		Attempt to write beyond EOF
247	IE.ONP	Hardware option not present
250	IE.DNA	Device not attached
251	IE.DAA	Device already attached
252	IE.DUN	Device not attachable
253	IE.RSU	Sharable resource in use
254	IE.OVR	Illegal overlay request
255	IE.BCC	Block check or CRC error
256	IE.NOD	Caller's nodes exhausted
257	IE.IFU	Index file full
260	IE.HFU	File header full
261	IE.WAC	Accessed for write
262	IE.CKS	File header checksum failure

(continued on next page)

Table G-5 (cont)

DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
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
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 
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

(continued on next page)

G-22 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-5 (cont)

DAP Miccode Values for Use with Maccode Values of 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
324	KSI	Index bucket will not hold two keys for key of reference
325	MBC	Multibuffer count invalid (negative value)
326	NET	Network operation failed at remote node
327	OK__ALK	Record is already locked
330	OK__DEL	Deleted record successfully accessed
331	OK__LIM	Retrieved record exceeds specified key value
332	OK__NOP	Key XAB not filled in
333	OK__RNF	Nonexistent record successfully accessed
334	PLV	Unsupported prologue version
335	REF	Illegal key of reference in XAB
336	RSL	Invalid resultant string length
337	RVU	Error updating RRVs; some paths to data may be lost
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

(continued on next page)

Table G-5 (cont)

DAP Miccode Values for Use with Maccode Values 0, 1, 4, 5, 6, 7

<i>Error Code (bits 0-11)</i>	<i>Corresponding Symbolic Status Code</i>	<i>Error Description</i>
360	TNS	Terminator not seen
361	BES	Bad escape sequence
362	PES	Partial escape sequence
363	WCC	Invalid wildcard context value
364	IDR	Invalid directory rename operation
365	STR	User structure (FAB/RAB) became invalid during operation
366	FTM	Network file transfer mode precludes operation
6000 to 7777		User-defined errors

Table G-6 follows. The message type number is contained in bits 0-11.

G-24 SUMMARY OF REMOTE FILE ACCESS ERROR/COMPLETION CODES

Table G-6
DAP Miccode Values for Use with Maccode Value 12

<i>Type Number (bits 0-11)</i>	<i>Message Type</i>
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

APPENDIX H

TASK-TO-TASK PROGRAMMING EXAMPLES

The following sections illustrate task-to-task communication for FORTRAN, COBOL, BASIC-PLUS-2, PASCAL, MACRO-11 and DLX. There are two examples for each language – a transmit and a receive program. Each program describes the activities of two cooperating tasks.

NOTE

The programming examples are also included on your installation diskette.

H.1 FORTRAN PROGRAMMING EXAMPLES

For both FORTRAN examples, FTNTRN is the transmit task and FTNREC is the receiver task. In the first example, FTNTRN accesses the network, connects to FTNREC, transmits inquiries to FTNREC, and finally receives responses from FTNREC. In the second example, FTNTRN completes sending inquiries to FTNREC, disconnects the link, deaccesses the network and then exits. Any error found by FTNREC is transmitted to FTNTRN as an interrupt message. FTNTRN then displays the interrupt message on the user's terminal.

H-2 TASK-TO-TASK PROGRAMMING EXAMPLES

H.1.1 FORTRAN Transmit Program

```
C
C   TO TASK BUILD USE THE FOLLOWING COMMAND STRING:
C
C   FTNTRN,FTNTRN=FTNTRN,[1,5]NETSUB/LB,F4POTS/LB,RMSLIB/LB
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                      (if RMS included)
C   //
C
C   INTEGER*2 MLTYP,RECSIZ,SNDSIZ,OPNLUN,CONLUN,MESNUM,XMITS,NDLEN,TSKLEN
C   INTEGER*2 IOST(2),MSTAT(3)
C   BYTE ERRMES(2),TSKNAM(6),CONBLK(72),NDNAM(6),DEFNOD(6),DEFTSK(6)
C   BYTE SNDBUF(50),RECBUF(10)
C   LOGICAL*1 STAT,IMMED
C   DATA DEFNOD/'M','A','S','T','E','R'/
C   DATA DEFTSK/'R','E','C','V','E','R'/
C
C   INITIALIZE CONSTANTS
C
C       IMMED=.TRUE.                      !* SET IMMED TO TRUE FOR GNDNTW
C       OPNLUN=1                          !* NETWORK OPNNT LUN
C       CONLUN=2                          !* COUNT LUN FOR THE
C                                         !* LOGICAL LINK
C       XMITS=20                          !* THE NUMBER OF INQUIRIES
C                                         !* TO BE SENT TO THE REMOTE NODE
C       SNDSIZ=50                         !* THE SIZE OF THE MESSAGES TO
C                                         !* BE SENT TO THE REMOTE NODE
C       RECSIZ=10                         !* THE SIZE OF THE MESSAGES TO
C                                         !* BE RECEIVED
C
C   GET THE NODE AND TASK NAMES
C
C       4      TYPE 300                      !* ASK FOR NODE NAME
C       READ(5,310) (NDNAM(NDLEN),NDLEN=1,6) !* GET THE NAME
C       DO 5 NDLEN=6,1,-1                  !* LOOP TO FIND LENGTH OF NAME
C       IF (NDNAM(NDLEN).NE.' ') GOTO 6 !* IF NOT A SPACE, GET TASK NAME
C       5      CONTINUE
C       DO 50 I=1,6
C       50     NDNAM(I)=DEFNOD(I)           !* DEFAULT NODE NAME 'MASTER'
C           NDLEN=6                        !* LENGTH OF DEFAULT NAME
C
C       6      TYPE 320                      !* ASK FOR THE TASK NAME
C       READ(5,310) (TSKNAM(TSKLEN),TSKLEN=1,6) !* GET IT
C       DO 7 TSKLEN=6,1,-1                  !* TSKLEN IS LENGTH OF TASK NAME
C       IF (TSKNAM(TSKLEN).NE.' ') GOTO 8 !* IF NOT SPACE, ACCESS NETWORK
C       7      CONTINUE
C       DO 60 I=1,6
C       60     TSKNAM(I)=DEFTSK(I)          !* DEFAULT TASK NAME 'RECV'
C           TSKLEN=6                      !* LENGTH OF DEFAULT NAME
C
C   ACCESS NETWORK
C
C       8      CALL    OPNNTW(OPNLUN,IOST,MSTAT)
C       IF      (IOST(1).NE.1)GOTO 100 !* IF FAILURE JUST EXIT
C
C   BUILD A FORMAT 2 CONNECT BLOCK
C
C       CALL    BFMT1(STAT,CONBLK,NDLEN,NDNAM,,TSKLEN,TSKNAM)
C       IF      (STAT)GOTO 10              !* IF SUCCESS GO ON
C       TYPE    200                        !* ELSE TYPE OUT A FAILURE
C                                         !* NOTIFICATION
C       GOTO    90                          !* AND EXIT
```

(continued on next page)

FORTTRAN Transmit Program (cont.)

```

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 INTERRUPT 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

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>: ',I3)
310     FORMAT    (6A1)
320     FORMAT    (' PLEASE ENTER TASK NAME <RECVER>: ',I3)
END

```

H-4 TASK-TO-TASK PROGRAMMING EXAMPLES

H.1.2 FORTRAN Receive Program

```
C
C   TO TASK BUILD USE THE FOLLOWING COMMAND STRING:
C
C   FTNREC,FTNREC=FTNREC,[1,5]NETSUB/LB,F4POTS/LB,RMSLIB/LB
C   /
C   UNITS=10
C   ACTFIL=4
C   EXTTSK=1000                      (if RMS included)
C   //
C
C   INTEGER*2 OPNLUN,MLTYP,INDEX,ACCLUN,NUMBER,NUMMES
C   INTEGER*2 RECSIZ,SNDSIZ,INTSIZ
C   INTEGER*2 MSTAT(3),IOST(2),IOST1(2),IOST2(2)
C   BYTE RECBUF(50),SNDDAT(10),MLBX(98),INTMES(2)
C
C   INITIALIZE CONSTANTS
C
C   OPNLUN=1                        !* NETWORK OPNNT LUN
C   ACCLUN=2                        !* ACCNT LUN FOR THE LOGICAL LINK
C   RECSIZ=50                       !* SIZE OF DATA BUFFER TO BE RECEIVED
C   INTSIZ=2                        !* SIZE OF INTERRUPT DATA BUFFER TO SEND
C   NUMMES=0                        !* NUMBER OF MESSAGES RECEIVED
C   SNDSIZ=10                       !* NUMBER OF BYTES TO SEND BACK
C
C   ACCESS NETWORK
C
C   CALL      OPNTW(OPNLUN,IOST,MSTAT)
C   IF        (IOST(1).NE.1)GOTO 100      !* IF FAILURE JUST EXIT
C   IF        (MSTAT(1).EQ.0)GOTO 40       !* IF NOTHING ON MAILBOX
C                                         !* 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
C   IF        (IOST1(1).NE.1)GOTO 40      !* IF GNDNT FAILED JUST
                                         !* CLOSE AND EXIT
C   IF        (MLTYP.GE.3)GOTO 40         !* IF MLTYP>=3 THE LINK HAS
                                         !* BEEN BROKEN
C   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
C   CALL      ACCNTW(ACCLUN,IOST,MLBX)
C   IF        (IOST(1).NE.1)GOTO 10      !* IF FAILURE ISSUE A NEW
                                         !* GNDNT
C
C   ISSUE A RECEIVE TO PICK UP DATA
C
C   30  CALL      RECNT(ACCLUN,IOST2,RECSIZ,RECBUF)
C       GOTO      10                      !* ISSUE A NEW GNDNT
                                         !* AND WAIT FOR A COMPLETION
C
C   WE COME HERE UPON RECEIVING A DISCONNECT OR ABORT
C
C   40  CALL      CLSNTW
C       GOTO      100                     !* DEACCESS THE NETWORK
                                         !* AND EXIT
C
C   WE COME HERE IF WE RECEIVE AN INQUIRY
C
C   50  NUMMES=NUMMES+1                   !* INCREMENT THE MESSAGE
```

(continued on next page)

FORTTRAN Receive Program (cont.)

```

C                                     !* 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

```

H-6 TASK-TO-TASK PROGRAMMING EXAMPLES

H.2 COBOL PROGRAMMING EXAMPLES

For both COBOL examples, COBTRN is the transmit task and COBREC is the receiver task. In the first example, COBTRN accesses the network, connects to COBREC, transmits inquiries to COBREC, and finally receives responses from COBREC. In the second example, COBTRN completes sending inquiries to COBREC, disconnects the link, deaccesses the network and then exits. Any error found by COBREC is transmitted to COBTRN as an interrupt message. COBTRN then displays the interrupt message on the user's terminal.

H.2.1 COBOL Transmit Program

IDENTIFICATION DIVISION.
PROGRAM-ID. COBTRN.

```
*****
*
*      THIS IS THE TRANSMIT PROGRAM OF THE DECNET COBOL
*      INTERFACE COMMUNICATION EXAMPLE PROGRAMS.
*
*      TO TASK BUILD, CREATE A FILE NAMED COBTRNBLD.CMD
*      CONTAINING THE FOLLOWING COMMAND STRING:
*
*      COBTRN,COBTRN=COBTRN,[1,5]NETSUB/LB,C81LIB/LB
*      /
*      LIBR=RMSRES:RO
*      UNITS=10
*      ACTFIL=4
*      EXTTSK=1000                      (if RMS included)
*      //
*
*      AND THEN TYPE THE FOLLOWING COMMAND STRING:
*
*      LINK @COBTRNBLD
*
*****
```

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).

(continued on next page)

COBOL Transmit Program (cont.)

WORKING-STORAGE SECTION.

01 MSGS.

03 MSG1.

05 FILLER PIC X(32) VALUE " NETWORK OPEN FAILED,
"IOST(1) = ".

05 MSG1-STAT1 PIC -99999.

05 FILLER PIC X(11) VALUE " IOST(2) = ".

05 MSG1-STAT2 PIC -99999.

03 MSG2.

05 FILLER PIC X(25) VALUE " CONNECT FAIL, IOST(1)
" = ".

05 MSG2-STAT1 PIC -99999.

05 FILLER PIC X(11) VALUE " IOST(2) = ".

05 MSG2-STAT2 PIC -99999.

03 MSG3.

05 FILLER PIC X(20) VALUE " ERROR ON INQUIRY # ".

05 MSG3-ERR1 PIC X(2).

03 MSG4.

05 FILLER PIC X(31) VALUE " ERROR ON INQUIRY DURI
"NG SEND: ".

05 MSG4-NUM1 PIC 99.

03 MSG5.

05 FILLER PIC X(34) VALUE " ERROR ON INQUIRY DURI
"NG RECEIVE: ".

05 MSG5-NUM1 PIC 99.

01 ARRAYS.

03 IOST.

05 IOSTAT OCCURS 2 TIMES PIC S9999 USAGE COMP.

03 MSTAT.

05 MSTATS OCCURS 3 TIMES PIC S9999 USAGE COMP.

01 STORE-STUFF.

03 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 XMIT5 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 ERMES 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.

(continued on next page)

H-8 TASK-TO-TASK PROGRAMMING EXAMPLES

COBOL Transmit Program (cont.)

```
*****
*
*      INPUT NODE NAME AND RECEIVER TASK NAME FROM
*      TERMINAL.
*
*****

      DISPLAY "ENTER NODE-NAME <MASTER>".
      ACCEPT IN-FILE.
      MOVE IN-FILE TO NODNAM.
      DISPLAY "ENTER TASK-NAME <RECVER>".
      ACCEPT IN-FILE.
      MOVE IN-FILE TO TSKNAM.

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

      CALL "OPNNTW" USING
          OPNLUN
          IOST
          MSTAT
          TEN.
      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 UNSUCCESSFULLY, PRINT AN ERROR MESSAGE
*      AND CLOSE THE NETWORK.  OTHERWISE, PRINT "LINK
*      ENABLED" MESSAGE.
*
*****
```

(continued on next page)

COBOL Transmit Program (cont.)

```

CALL "CONNTW" USING
    CONLUN
    IOST
    CONBLK.
IF IOSTAT(1) = 1
    NEXT SENTENCE
ELSE
    MOVE SPACES TO RESULT-REC
    MOVE IOSTAT(1) TO MSG2-STAT1
    MOVE IOSTAT(2) TO MSG2-STAT2
    MOVE MSG2 TO RESULT-REC
    DISPLAY RESULT-REC
    GO B100-CLOSE.
DISPLAY "LINK ENABLED".

*****
*
*   SEND AND RECEIVE MESSAGES FROM THE REMOTE NODE.
*   IF THERE IS SOMETHING ON THE NETWORK DATA QUEUE
*   (MSTATS (1) > 0), GET THE MESSAGE.
*
*****

LOOP.   PERFORM LOOP VARYING MESNUM FROM 1 BY 1 UNTIL MESNUM = XMITS.
        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.

```

(continued on next page)

H-10 TASK-TO-TASK PROGRAMMING EXAMPLES

COBOL Transmit Program (cont.)

```
*****
*
*      RECEIVE A MESSAGE FROM THE REMOTE NODE.  IF
*      UNSUCCESSFUL, PRINT AN ERROR MESSAGE AND START
*      THE NEXT TRANSMISSION.  IF SUCCESSFUL, SIMPLY
*      START THE NEXT TRANSMISSION.
*
*****

      CALL "RECNTW" USING
          CONLUN
          IOST
          RECSIZ
          RECBUF.
      IF IOSTAT(1) = 1
          NEXT SENTENCE
      ELSE
          MOVE SPACES TO RESULT-REC
          MOVE MESNUM TO MSG5-NUM1
          MOVE MSG5 TO RESULT-REC
          DISPLAY RESULT-REC.

*****
*
*      DEACCESS THE NETWORK.
*
*****

B000-ENDLOOP.
      DISPLAY "LINK DISABLED".
      CALL "DSCNTW" USING
          CONLUN
          IOST.

*****
*
*      CLOSE THE NETWORK AND EXIT.
*
*****

B100-CLOSE.
      CALL "CLSNTW".
      DISPLAY "END OF EXECUTION".
C000-END.
      STOP RUN.
```

H.2.2 COBOL Receive Program

IDENTIFICATION DIVISION.
PROGRAM-ID. COBREC.

```
*****
*
*      THIS IS THE RECEIVE PROGRAM OF THE DECNET COBOL
*      INTERFACE COMMUNICATION EXAMPLE PROGRAMS.
*
*      TO TASK BUILD, CREATE A FILE NAMED COBRECBLD.CMD
*      CONTAINING THE FOLLOWING COMMAND STRING:
*
*      COBREC,COBREC=COBREC,[1,5]NETSUB/LB,C81LIB/LB
*      /
*      LIBR=RMSRES:RO
*      UNITS=10
*      ACTFIL=4
*      EXTTSK=1000                      (if RMS included)
*      //
*
*      AND THEN TYPE THE FOLLOWING COMMAND STRING:
*
*      LINK @COBRECBLD
*
*****
```

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.
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.

(continued on next page)

H-12 TASK-TO-TASK PROGRAMMING EXAMPLES

COBOL Receive Program (cont.)

```

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.
*
*****

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.

```

(continued on next page)

COBOL Receive Program (cont.)

```

*****
*
*      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.
*
*****

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.

```

(continued on next page)

H-14 TASK-TO-TASK PROGRAMMING EXAMPLES

COBOL Receive Program (cont.)

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

H.3 BASIC-PLUS-2 PROGRAMMING EXAMPLES

For both BASIC-PLUS-2 examples, BASTRN is the transmit task and BASREC is the receiver task. In the first example, BASTRN accesses the network, connects to BASREC, transmits inquiries to BASREC, and finally receives responses from BASREC. In the second example, BASTRN completes sending inquiries to BASREC, disconnects the link, deaccesses the network and then exits. Any error found by BASREC is transmitted to BASTRN as an interrupt message. BASTRN then displays the interrupt message on the user's terminal.

H.3.1 BASIC-PLUS-2 Transmit Program

```

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:[1,5]NETSUB/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 REENTER"                          &
      \ 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 REENTER"                          &
      \ PRINT \ GOTO 30

40      !!! DEFINE CONSTANTS !!!                                           &
      IMMED%=-1%                  !SET IMMED TO TRUE FOR GNDNTW           &
      \ OPNLUN%=1%                 !NETWORK OPNNT LUN                     &
      \ CONLUN%=2%                 !CONNT LUN FOR THE LOGICAL LINK        &
      \ XMITS%=20%                 !THE NUMBER OF INQUIRIES               &
      \                               !TO 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              &

```

(continued on next page)

BASIC-PLUS-2 Transmit Program (cont.)

```

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 CONFIRMATION
                                           !TO NETWORK

90      !!! SEND AND RECEIVE MESSAGES TO AND FROM THE REMOTE NODE !!!
FOR      MESNUM%=1% TO XMIT$

100      !!! FIRST GET ANY ERROR MESSAGES SENT FROM THE OTHER !!!
      !!! SIDE VIA INTERRUPT MESSAGES !!!
      IF MSTAT%(0%)=0% THEN 110 !IF MSTAT%(0%)=0% NO MESSAGES
                                           !ARE THERE
      ELSE CALL GNDNTW BY REF(IOST%(),MLTYP%,2%,ERRMES$
           ,DUMMY%,IMMED%,2%) !GET THE MESSAGE
\      IF IOST%(0%)<>1% THEN 110 !IF WE COULDN'T GET MESSAGE
                                           !JUST IGNORE IT
      ELSE PRINT "ERROR ON INQUIRY #";ASCII(LEFT(ERRMES$,1%))
                                           !PRINT OUT THE MESSAGE

110      !!! SEND THE INQUIRY !!!
CALL      SNDNTW BY REF(CONLUN%,IOST%(),SND$SIZE%,SND$BUF$)
\      IF IOST%(0%)=1% THEN 120!IF SUCCESS CONTINUE
      ELSE PRINT "ERROR ON INQUIRY DURING SEND: ";MESNUM%
                                           !OTHERWISE TYPE OUT AN ERROR
\      GOTO     130      !MESSAGE AND START A NEW MESSAGE

120      !!! RECEIVE THE RESPONSE FROM THE REMOTE NODE !!!
CALL      RECNTW BY REF(CONLUN%,IOST%(),RECS$IZE%,REC$BUF$)
\      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

```

H.3.2 BASIC-PLUS-2 Receive Program

```

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:[1,5]NETSUB/LB                            !!! &
      !!!      to the ODL file.                                           !!! &

      !!! INITIALIZE CONSTANTS !!!                                         &
      DIM MSTAT%(2%),IOST%(1%),IOST1%(1%),IOST2%(1%)                      &
      \ INTMESS$=STRING$(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

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

```

(continued on next page)

BASIC-PLUS-2 Receive Program (cont.)

```

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%(),SNSDSIZ%,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

```

H.4 PASCAL PROGRAMMING EXAMPLES

For both PASCAL examples, PASTRN is the transmit task and PASREC is the receiver task. In the first example, PASTRN accesses the network, connects to PASREC, transmits inquiries to PASREC, and finally receives responses from PASREC. In the second example, PASTRN completes sending inquiries to PASREC, disconnects the link, deaccesses the network and then exits. Any error found by PASREC is transmitted to PASTRN as an interrupt message. PASTRN then displays the interrupt message on the user's terminal.

H.4.1 PASCAL Transmit Program

```
PROGRAM PASTRN(INPUT, OUTPUT);

%INCLUDE 'LB:[1,5]NETDEFS.PAS/NOLIST'

LABEL
    90;

VAR
    MLTYP: INTEGER; (* Message type received by GNDNT *)
    MESNUM: INTEGER; (* Inductive message number *)
    IOST: NET_BLOCK_STATUS; (* Status block for network operations *)
    MSTAT: ARRAY [1..3] OF INTEGER; (* Message queue status *)
    ERRMES: PACKED ARRAY [1..2] OF BYTE; (* Buffer for receiver error
                                           message *)
    CONBLK: Connect_Block; (* Connect block for logical link *)
    NDNAME: PACKED ARRAY [1..6] OF CHAR; (* Node name *)
    NDLEN: INTEGER; (* Length of node name *)
    TSKNAM: PACKED ARRAY [1..6] OF CHAR; (* Task name *)
    TSKLEN: INTEGER; (* Length of task name *)
    SNDBUF: PACKED ARRAY [1..50] OF CHAR; (* Xmit data buffer *)
    RECBUF: PACKED ARRAY [1..10] OF CHAR; (* Recv data buffer *)
    STAT: INTEGER; (* Status info for BFMT1 *)

CONST
    DEFNOD = 'MASTER'; (* Default node name *)
    DEFTSK = 'RECVER'; (* Default task name *)

    IMMED = -1; (* set immed to true for GNDNT *)

    (* Use large numbers for LUNs as Pascal dynamically allocates them *)
    (* starting at 1 for each open file. The likelihood of having 25 *)
    (* concurrently open files is minimal. The system will have run *)
    (* out of memory long before that level is reached. *)

    OPNLUN = 25; (* network open lun *)
    CONLUN = 26; (* 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 message to be sent to the remote node *)
    RECSIZ = 10; (* the size of the message to be received *)
```

(continued on next page)

H-20 TASK-TO-TASK PROGRAMMING EXAMPLES

PASCAL Transmit Program (cont.)

```
BEGIN

  (* get the node and task names *)

  (* Ask, and you shall receive the node name *)

  WRITE('Please enter node name <MASTER>: ');
  READLN(NDNAME);

  (* Count down the trailing spaces to determine the actual name length *)

  NDLEN := 6;
  WHILE (NDNAME[NDLEN] = ' ') AND (NDLEN > 0) DO NDLEN := NDLEN - 1;

  (* If no name specified, use the default *)

  IF NDLEN = 0

  THEN
    BEGIN
      NDNAME := DEFNOD;
      NDLEN := 6;
    END;

  (* Ask, and you shall receive the task name *)

  WRITE('Please enter task name <RECVER>: ');
  READLN(TSKNAM);

  (* Count down the trailing spaces to determine the actual name length *)

  TSKLEN := 6;
  WHILE (TSKNAM[TSKLEN] = ' ') AND (TSKLEN > 0) DO TSKLEN := TSKLEN - 1;

  (* If no name specified, use the default *)

  IF TSKLEN = 0
  THEN
    BEGIN
      TSKNAM := DEFTSK;
      TSKLEN := 6;
    END;

  (* Access the network *)

  OPNNTW(OPNLUN, IOST, MSTAT);

  (* Only continue on success *)

  IF IOST[1] = 1
  THEN
    BEGIN
      (* Build a format 1 connect block *)

      BFMT1(STAT, CONBLK, NDLEN, NDNAME, , TSKLEN, TSKNAM);
      (* If unsuccessful, complain and quit *)

      IF NOT ODD(STAT)

      THEN
        BEGIN
          WRITELN('Error building connect block.');
```

```
          GOTO 90;
        END;

      (* Connect to the task on the remote node *)
```

(continued on next page)

PASCAL Transmit Program (cont.)

```

CONNTW(CONLUN, IOST, CONBLK);

(* Again, if unsuccessful, complain and quit *)

IF IOST[1] <> 1
  THEN
    BEGIN
      WRITELN('Connect fail, IOST= ', IOST[1], ',', IOST[2]);
      GOTO 90;
    END;

(* Announce successful initialization *)

WRITELN('Link enabled.');
```

(* Loop for several transactions *)

```

FOR MESNUM := 1 TO XMITS DO
  BEGIN

    (* If no message in queue, don't get a message from queue *)

    IF MSTAT[1] <> 0
      THEN
        BEGIN

          (* Get a message from the network *)

          GNDNTW(IOST, MLTYP, 2, ERRMES, , IMMED, 2);

          (* Complain about errors *)

          IF IOST[1] <> 1
            THEN WRITELN('Error on inquiry ', ERRMES[1]: 3);
          END;

          (* Send a request for another message *)

          SNDNTW(CONLUN, IOST, SNDSIZ, SNDBUF);

          (* Complain about errors *)
          (* If no error, receive a message *)

          IF IOST[1] <> 1
            THEN WRITELN('Error on inquiry ', MESNUM: 3)
            ELSE
              BEGIN

                (* Receive a message *)

                RECNTW(CONLUN, IOST, RECSIZ, RECBUF);

                (* Complain about errors *)

                IF IOST[1] <> 1
                  THEN WRITELN('Error on inquiry ', MESNUM: 3);
                END;
              END;
            END;

          (* Disconnect the network link *)

```

(continued on next page)

PASCAL Transmit Program (cont.)

```
        DSCNTW(CONLUN, IOST);  
        (* Announce the link termination *)  
        WRITELN('Link disabled');  
90:    (* Close the network session *)  
        CLSNTW;  
END;  
  
        (* Announce termination and quit *)  
        WRITELN('End of program execution');  
END.
```

H.4.2 PASCAL Receive Program

```

PROGRAM PASREC(Input, Output);

%INCLUDE 'LB:[1,5]NETDEFS.PAS/NOLIST'

CONST

    (* Use large numbers for LUNs as Pascal dynamically allocates them *)
    (* starting at 1 for each open file. The likelihood of having 25 *)
    (* concurrently open files is minimal. The system will have run *)
    (* out of memory long before that level is reached. *)

    OPNLUN = 25; (* Network OPNNT LUN *)
    ACCLUN = 26; (* ACCNT LUN for the logical link *)

    RECSIZ = 50; (* Size of data buffer to be received *)
    INTSIZ = 2;  (* Size of interrupt data buffer to send *)
    SNDSIZ = 10; (* Number of bytes to send back *)

VAR
    MLTYP: INTEGER; (* Message type code *)
    INDEX: INTEGER; (* Completion indicator for WAITNT *)
    NUMMES: INTEGER; (* Received message counter *)
    MSTAT: ARRAY [1..3] OF INTEGER; (* Message queue status *)
    IOST, IOST1, IOST2: ARRAY [1..2] OF INTEGER; (* Status blocks for network
                                                    operations *)
    RECBUF: PACKED ARRAY [1..50] OF CHAR; (* Receive data buffer *)
    SNDDAT: PACKED ARRAY [1..10] OF CHAR; (* Xmit data buffer *)
    MLBX: GND Block; (* Mail box for GNDNT data *)
    INTMES: PACKED ARRAY [1..1] OF INTEGER; (* Interrupt message buffer *)
    FLAG, DONE: BOOLEAN; (* Flags for control of program flow *)

BEGIN

    (* init the message counter *)

    NUMMES := 0;

    (* Access the network *)

    OPNNTW(OPNLUN, IOST, MSTAT);

    (* If no errors, go on *)

    IF IOST[1] = 1
    THEN
        BEGIN
            (* Init some flags *)

            FLAG := TRUE;
            DONE := MSTAT[1] = 0;

            (* Loop until error or disconnect *)

            WHILE NOT DONE DO
                BEGIN
                    (* If there is a message outstanding, read it *)

```

(continued on next page)

PASCAL Receive Program (cont.)

```

IF FLAG
  THEN GNDNT(IOST1, MLTYP, SIZE(GND_BLOCK), MLBX);

(* Wait for something to happen *)

WAITNT(INDEX, IOST1, IOST2);

(* If it happened to IOST2, a receive has completed *)

IF (INDEX = 2)
  THEN
    BEGIN

      (* Bump the message number *)

      NUMMES := NUMMES + 1;

      IF IOST2[1] = 1
        THEN
          BEGIN

            (* if no error, send back the data *)

            SNDNTW(ACCLUN, IOST, SNDSIZ, SNDDAT)

          END
        ELSE
          BEGIN

            (* If error, send an interrupt *)

            INTMES[1] := NUMMES;
            XMINT(ACCLUN, IOST, INTSIZ, INTMES);

          END;

            (* Issue another receive request *)

            RECNT(ACCLUN, IOST, RECSIZ, RECBUF);

            (* Set flag for no GNDNT *)

            FLAG := FALSE;

          END
        ELSE IF (IOST1[1] = 1) AND (MLTYP < NT_DSC)
          THEN
            BEGIN

              (* If it happened to IOST1, Network data has been
              received *)

              (* If it was a connect request, accept it *)

              IF MLTYP = NT_CON
                THEN
                  BEGIN
                    ACCNTW(ACCLUN, IOST, MLBX);

                    (* If no errors, issue a receive data and wait

```

(continued on next page)

PASCAL Receive Program (cont.)

```
                some more *)
                IF IOST[1] = 1
                THEN RECNT(ACCLUN, IOST2, RECSIZ, RECBUF);
                END;

                (* We processed a GNDNT, set FLAG so another one will
                be issued *)

                FLAG := TRUE;
            END
        ELSE

            (* An error occurred, set DONE and quit *)

            DONE := TRUE;
        END;

        (* Close the network prior to shutting down *)

        CLSNTW;
    END;

    (* Announce the shut down, and quit *)

    WRITELN('End of execution');
END.
```


H.5 MACRO-11 PROGRAMMING EXAMPLES

For both MACRO-11 examples, SEN10 is the transmit task and REC10 is the receiver task. In the first example, SEN10 successfully transmits 10 data messages to the cooperating program named REC10. Both programs disconnect from the network once REC10 successfully receives all ten messages. In the second example, each time that REC10 receives a message from SEN10, "THIS IS MESSAGE n" is displayed on the console device. The actual data message is immediately delivered to REC10 as an interrupt message.

H.5.1 MACRO-11 Transmit Program

```

                                .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 using the Tool Kit, create a file named
; SEN10ASM.CMD containing the following command string:
;
;   SEN10,SEN10/-SP=[1,5]NETLIB/ML,SEN10
;
; and then type the following command string:
;
;   MAC @SEN10ASM
;
; To task build using the Tool Kit, create a file named
; SEN10BLD.CMD containing the following command string:
;
;   SEN10,SEN10/-SP=SEN10,[1,5]NETLIB/LB
;
; and then type the following command string:
;
;   LINK @SEN10BLD
;*****
;
; .MCALL  OPNW$$,CONW$$,SNDW$$,CONB$$,ALUN$$,QIOW$$
; .MCALL  EXIT$$,MRKT$$,WTSE$$,CLEF$$,SETF$$,QIO$$
; .MCALL  DSCW$$,XMIW$$,ASTX$$
;
; DATA AREA
;
; MESH:  .ASCII  /THIS IS MESSAGE /      ; MESSAGE TO BE TRANSMITTED
NUM:     .ASCII  /0/                      ; MESSAGE NUMBER
NN=. -MESH
PRMPT:   .ASCII  /MSG:/                    ; PROMPT FOR INTERRUPT MESSAGE
; .EVEN
IOSTN:   .BLKW   2                          ; COMPLETION STATUS FOR NETWORK
BUFF:    .BLKB   16.                        ; INTERRUPT MESSAGE BUFFER
IOSTB:   .BLKW   2                          ; COMPLETION STATUS FOR BUFFER
CNT:     .WORD   0                          ; NUM OF CHAR IN INTERRUPT MESS
ERRCNT:  .WORD   0                          ; ERROR COUNT
IOSB:    .BLKW   1                          ; I/O STATUS
;
; .EVEN
CONBL:   CONB$$  ELROND,0,1,<REC10>          ; CONNECT REQUEST BLOCK
;
; CODE
;
; .EVEN

```

(continued on next page)

MACRO-11 Transmit Program (cont.)

```

START:  CLR      ERRCNT          ; INITIALIZE ERROR COUNT TO ZERO
        CLEF$C  5              ; CLEAR EVENT FLAG USED TO MAKE SURE
                                ; INTERRUPT MESSAGE ACCEPTED PRIOR
                                ; TO EXIT
                                ; INITIALIZE MESSAGE NUM TO ZERO
        MOVB    #60,NUM         ; ASSIGN LUN 1 FOR NETWORK DATA QUEUE
        ALUN$C  1,NS           ; ASSIGN LUN 2 FOR LOGICAL LINK
        ALUN$C  2,NS           ; CREATE THE NETWORK DATA QUEUE
        OPNW$S  #1,#1,#IOSTN    ; TEST FOR ERRORS
        TSTB    IOSTN

        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]
                                ; TEST FOR ERRORS
        TST     $DSW
        BLT     ERR3
        MOV     #10.,R0        ; SET LOOP COUNTER TO 10
LOOP:   SNDW$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$S
;
;   TERMINAL AST ROUTINE
;
TRMAST: MOV     (SP)+,IOSB       ; POP STACK
        MOV     IOSTB+2,CNT      ; OBTAIN NUMBER OF CHARACTERS
        XMIW$S  #2,#3,#IOSTN, ,<#BUFF,CNT>; TRANSMIT INTERRUPT MESSAGE
                                ; (NOTE USE OF EF 3 INSTEAD OF
                                ; EF 2 - AVOID COMPETITION)
                                ; TEST FOR ERRORS
        TSTB    IOSTN
        BLE     ERR5
        SETF$C  5              ; SET EVENT FLAG TO INDICATE
                                ; INTERRUPT MESSAGE SENT
        ASTX$S
;
;   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

```

H.5.2 MACRO-11 Receive Program

```

.TITLE REC10
;*****
;
; THIS EXAMPLE WILL:
;   ACCEPT SHORT MESSAGES FROM THE SENDER TASK "SND10"
;   PRINT THE MESSAGES ON THE CONSOLE DEVICE (C0:)
;   DISCONNECT AND EXIT GRACEFULLY.
;
;
;   To assemble using the Tool Kit, create a file named
;   REC10ASM.CMD containing the following command string:
;
;       REC10,REC10/-SP=[1,5]NETLIB/ML,REC10
;
;   and then type the following command string:
;
;       MAC @REC10ASM
;
;   To task build using the Tool Kit, create a file named
;   REC10BLD.CMD containing the following command string:
;
;       REC10,REC10/-SP=REC10,[1,5]NETLIB/LB
;
;   and then type the following command string:
;
;       LINK @REC10BLD
;*****
.MCALL OPNW$$,SPAW$$,RECW$$,GNDW$$,ACCW$$,CLSW$$,NETDF$
.MCALL QIOW$$,ALUN$$,CLEF$$,WTSE$$,SETF$$,ASTX$$,EXIT$$
NETDF$
;
; DATA AREA
;
BUF1:  .BLKB   25.                ; BUFFER FOR USER MESSAGES
      .EVEN
BUF2:  .BLKB   N.CBL              ; BUFFER FOR NETWORK MESSAGES
IOST:  .BLKW   2                  ; COMPLETION STATUS FOR NETWORK
IOST1: .BLKW   2                  ; COMP. STAT. FOR GET NET DATA
IOST2: .BLKW   2                  ; COMP. STAT. FOR ACCEPT CONNECT
IOSB:  .BLKW   1                  ; I/O STATUS
ERRCNT: .WORD   0                 ; ERROR COUNT
CNT:   .WORD   0                 ; USER MESSAGE CHAR COUNT
CNTB:  .BLKB   2                  ; INTERRUPT MESSAGE CHAR COUNT
FLAG:  .WORD   0                 ; DISCONNECT FLAG
      .EVEN
;
; CODE
;
START: CLR      ERRCNT            ; INITIALIZE ERROR COUNT TO ZERO
;      CLEF$$  10.                ; CLEAR EVENT FLAG USED TO MAKE
;                                   ; SURE CONNECT HAS OCCURRED
      ALUN$$  1,NS                ; ASSIGN LUN 1 FOR NETWORK DATA QUEUE
      ALUN$$  2,NS                ; ASSIGN LUN 2 FOR LOGICAL LINK
      OPNW$$  #1,#1,#IOST         ; CREATE THE NETWORK DATA QUEUE
      TSTB    IOST                ; TEST FOR ERRORS
      BLE     ERR1
      SPAW$$  #1,#1,#IOST,#CMPAST,<#NETAST> ; SPECIFY AST HANDLING
      TSTB    IOST                ; TEST FOR ERRORS
      BLE     ERR2

```

(continued on next page)

MACRO-11 Receive Program (cont.)

```

        WTSE$C 10.                                ; WAIT TO MAKE SURE CONNECT
                                                ; HAS OCCURRED
LOOP:   RECWS$S #2,#2,#IOST,,<#BUF1,#25.>; RECEIVE UP TO 25 CHARS
        TSTB    IOST                                ; TEST FOR ERRORS
        BLE     ERR3
        MOV     IOST+2,CNT                          ; OBTAIN CHARACTER COUNT
        QIOW$S  #IO.WLB,#5,#5,,,<#BUF1,CNT,#40>; TYPE MESSAGE
                                                ; ON TERMINAL
        TST     FLAG                                ; HAS DISCONNECT OCCURRED?
        BEQ     LOOP                                ; NO, POST ANOTHER RECEIVE
        CLSW$S  #1,#1,#IOST2                        ; CLOSE NETWORK
        TSTB    IOST2                                ; TEST FOR ERRORS
        BLE     ERR5
        EXIT$S                                     ; PROGRAM EXIT
        BR      LOOP

;
;  ERROR HANDLING - A SAMPLE DEBUGGING TECHNIQUE
;
ERR6:   INC     ERRCNT
ERR5:   INC     ERRCNT
ERR4:   INC     ERRCNT
ERR3:   INC     ERRCNT
ERR2:   INC     ERRCNT
ERR1:   INC     ERRCNT
        MOV     ERRCNT,R1                          ; R1 = ERROR NUMBER
        MOV     $DSW,R2                            ; R2 = DIRECTIVE STATUS WORD
        MOV     IOST,R3                            ; R3 = I/O STATUS BLOCK (1ST WORD)
        MOV     IOST+2,R4                          ; R4 = I/O STATUS BLOCK (2ND WORD)
        IOT                                         ; ABORT - DUMP REGISTERS

;
;
;  AST HANDLING FOR DATA IN NETWORK DATA QUEUE
;
CMPAST: MOV     (SP)+,IOSB                          ; SAVE SPA$ I/O STATUS BLOCK ADDR
        MOV     R0,-(SP)                            ; SAVE R0
        MOV     IOSB,R0                            ; GET I/O STATUS BLOCK ADDRESS
        CMPB    #IS.SUC,(R0)                        ; SUCCESSFUL?
        BEQ     OKA
        JMP     OUT
OKA:    MOV     2(R0),R0                            ; GET CURRENT NETWORK DATA COUNT
        BNE     OKB
        JMP     OUT
OKB:    BR      GET
NETAST: MOV     R0,-(SP)                            ; SAVE R0
        MOV     #1,R0                              ; SET NETWORK DATA COUNT TO 1
GET:    GNDW$S  #1,#1,#IOST1,,<#BUF2,#N.CBL>; GET NETWORK DATA
        BCS     OUT                                ; CARRY BIT SET - ERROR
        CMPB    #IS.SUC,IOST1                      ; SUCCESSFUL?
        BNE     OUT
        CMPB    #NT.CON,IOST1+1                    ; CHECK IF CONNECT REQUEST
        BNE     OTHER
        ACCW$S  #2,#2,#IOST2,,<#BUF2>; ACCEPT CONNECTION
        TSTB    IOST2                                ; TEST FOR ERRORS
        BLE     ERR4
        SETF$C 10.                                ; SET EVENT FLAG TO INDICATE
                                                ; CONNECT 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

```

(continued on next page)

MACRO-11 Receive Program (cont.)

```

;
;
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$$  #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$$  ; AST EXIT

;
;
;
        .END      START

```

H.6 DLX QIO PROGRAMMING EXAMPLES

Both DLX examples use DLX QIOs to transmit and receive data over the Ethernet. In the first example, the transmit task called XTS sends data across the Ethernet. In the second example, the receiver task called XTR echoes back the received data over the Ethernet.

H.6.1 DLX Transmit Program

```
.TITLE  XTS - DLX TRANSMITTER
.IDENT  /V01.00/

;
; COPYRIGHT (C) 1983,1984 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  12-JAN-84
;       PRO/DECnet v1.0
;
;
;+
;
; XTS - DLX SYSTEM EXERCISER (THIS TEST UTILITY IS UNSUPPORTED)
;
; XTS IS A UTILITY WHICH ENABLES A USER TO TRANSMIT DATA READ IN FROM
; A TERMINAL OR COMMAND FILE ACROSS THE ETHERNET TO A RECEIVER TASK WHICH
; ECHOES THE RECEIVED DATA BACK OVER THE ETHERNET. XTS USES THE DLX INTERFACE
; TO PROVIDE ACCESS TO THE ETHERNET.
;
;
; TO ASSEMBLE USING THE TOOL KIT, USE THE FOLLOWING COMMAND STRING:
;
;     PMA XTS,XTS/-SP=LB:[1,5]NETLIB/ML,XTS
;
;
```

(continued on next page)

DLX Transmit Program (cont.)

```

;      TO TASK BUILD USING THE TOOL KIT, USE THE FOLLOWING COMMAND STRING:
;
;      XTS/PR:0,XTS/-SP=XTS,LB:[1,5]NETLIB/LB:GCL
;      /
;      STACK=30
;      UNITS=4
;      ASG=TI:1:2:3:4
;      TASK=...XTS
;      //
;
; THE FOLLOWING IS AN EXAMPLE OF THE XTS DIALOG:
;
;      >XTS
;      LINE: CNA-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: CNA-0
;
; WHEN FINISHED WITH XTS/XTR, XTR MUST BE ABORTED.  NOTE - YOU MUST BE IN
; THE TOOL KIT/DCL IN ORDER TO ABORT XTR WITHOUT CAUSING YOUR PRO TO
; BUGCHECK WITH A 300/6 ERROR!
;-

.SBTTL  LOCAL MACROS

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

.SBTTL  MACRO CALLS

.MCALL  QIOW$,QIO$,QIOW$$,ALUN$$,EXIT$$,EXST$$,FSRSZ$,ASTX$$
.MCALL  GCL$,GCLDF$,CALLR,DLXDF$,EPMDF$

DLXDF$          ; DEFINE DLX FUNCTION CODES
EPMDF$          ; DEFINE ADDITIONAL VALUES FOR
                ;   ETHERNET

.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

```

(continued on next page)

DLX Transmit Program (cont.)

```

; 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

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

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

;
; SET CHARACTERISTICS BUFFER
;
SETCHR: .WORD    CC.DST                                ;SETTING UP PROTOCOL TYPE/ADDRESS
        .WORD    10.                                ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD    0                                  ;RESERVED FIELD
        .WORD    0                                  ;STATUS WORD
        .WORD    1                                  ;DEFINE PROTOCOL TYPE
        .WORD    LF$PAD                              ;PAD MESSAGES TO MINIMUM LENGTH
        .BYTE    252                                ;DEFINE XTR ADDRESS (DECNET ADDRESS)
        .BYTE    0
        .BYTE    4
        .BYTE    0
        .BYTE    231
        .BYTE    20
SETLEN=.-SETCHR                                ;LENGTH OF SET CHAR BUFFER.

;
; PROTOCOL/ADDRESS BUFFER FOR TRANSMITTING
;
XMICHR: .WORD    CC.ADR                                ;THIS PART DEFINES AN ADDRESS
        .WORD    6                                  ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD    0                                  ;RESERVED FIELD
        .WORD    0                                  ;STATUS WORD
        .BYTE    252                                ;DEFINE XTR ADDRESS (DECNET ADDRESS)
        .BYTE    0
        .BYTE    4
        .BYTE    0
        .BYTE    231
        .BYTE    20

```

(continued on next page)

DLX Transmit Program (cont.)

```

        .WORD    CC.PRO           ;THIS PART DEFINES A PROTOCOL TYPE
        .WORD    2                ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD    0                ;RESERVED WORD
        .WORD    0                ;STATUS WORD
        .WORD    1                ;PROTOCOL TYPE = 1
XMILEN=-XMICHR                      ;LENGTH OF WHOLE XMICHR BUFFER

;
; 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
SFTErr: .ASCII   /Error Defining Protocol-Address -- /
BUFSET:  .BLKB    7
XMTERR:  .ASCII   /Error transmitting data -- /
BUFXTMT: .BLKB    7
RECERR:  .ASCII   /Error receiving data -- /
BUFREC:  .BLKB    7
        .LIST    BEX
        .EVEN

        .SBTTL   XTS -- XTS MAIN LINE

;+
; XTS -- MAIN LINE OF XTS CODE
;-

```

(continued on next page)

DLX Transmit Program (cont.)

```

XTSEP::
        MOV     #EX$SUC,EXSTAT           ;ASSUME EXIT WITH STATUS
;
; ASSIGN LUN TO CHANNEL
;
        ALUN$$  #CHNLUN,#"NX,#0
        BCC     10$                      ;IF CC, ALL OKAY
        EPRINT  DLXERR                    ;ELSE, ASSUME DLX NOT LOADED
        BR      EXIT                      ;AND LEAVE
;
; PROMPT USER FOR LINE ID
;
10$:    MOV     $CLPMT,-(SP)               ;SAVE CURRENT PROMPT
        MOV     #PROMPT,$CLPMT           ;PROMPT STRING
        CALL    GCL                      ;GET A COMMAND LINE
        MOV     (SP)+,$CLPMT             ;RESTORE PROMPT
        BCS     EXIT                     ;IF CS, ASSUME EOF
        TST     R5                       ;BLANK LINE ?
        BEQ     10$                     ;IF EQ, YES - TRY AGAIN
;
; OPEN ACCESS TO THE LINE.
;
        QIOW$$  #IO.XOP,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5>
        BCS     15$                      ; IF CS, ERROR
        MOVB    CHNSB,R1                 ; SUCCESSFUL ?
        BPL     16$                      ; 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
;
; DEFINE THE ETHERNET ADDRESS AND PROTOCOL TYPE OF THE XTR PROGRAM
;
16$:    QIOW$$  #IO.XSC,#CHNLUN,#CHNEFN,,#CHNSB,,<#SETCHR,#SETLEN>
        BCS     17$                      ; IF CS, ERROR
        MOVB    CHNSB,R1                 ; SUCCESSFUL ?
        BPL     20$                      ; IF PL, YES
        MOV     #BUFSET,R0               ; ELSE, GET BUFFER ADDRESS
        CLR     R2                      ; ZERO SUPPRESSION
        CALL    $CBOMG                   ; CONVERT NUMBER
        CLRB    (R0)                    ; MAKE STRING ASCIZ
17$:    EPRINT  SETERR                    ; 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)

DLX Transmit Program (cont.)

```

;
; 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)

DLX Transmit Program (cont.)

```

        .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$$  #IO.XMT,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5,#XMICHR,#XMILEN>
        BCS     10$                     ;IF CS, ERROR
        MOVB    CHNSB,R1                 ;SUCCESSFUL ?
        BPL     20$                     ;IF PL, YES
        MOV     #BUFXT,R0                ;ELSE, GET BUFFER ADDRESS
        CLR     R2                       ;ZERO SUPPRESSION
        CALL    $CBOMG                  ;CONVERT NUMBER
        CLRB    (R0)                    ;MAKE STRING ASCIZ
10$:    EPRINT  XMTERR                  ;TRANSMIT ERROR
        SEC                      ;INDICATE FAILURE
20$:    RETURN

        .SBTTL  RECAST - AST FOR CHANNEL READ COMPLETE

```

(continued on next page)

DLX Transmit Program (cont.)

```

;+
; **--RECAST - AST FOR CHANNEL READ COMPLETE
;
; INPUTS:
;   (SP) = ADDRESS OF I/O STATUS BLOCK
;
; OUTPUTS:
;   1. ANOTHER READ HUNG ON CHANNEL (IF LAST RECEIVE SUCEEDED)
;   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$$

      .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)

DLX Transmit Program (cont.)

```

        .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

```

H-40 TASK-TO-TASK PROGRAMMING EXAMPLES

H.6.2 DLX Receive Program

```
.TITLE  XTR - DLX RECEIVER
.IDENT  /V01.00/

;
; COPYRIGHT (C) 1983,1984 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  12-JAN-84
;       PRO/DECnet V1.0
;
;
;+
;
; XTR - DLX SYSTEM EXERCISER (THIS TEST UTILITY IS UNSUPPORTED)
;
; XTR ECHOS RECEIVED DATA BACK OVER THE ETHERNET. XTR USES THE DLX UTILITY
; TO PROVIDE THE ACCESS TO THE ETHERNET.
;
;
; TO ASSEMBLE USING THE TOOL KIT, USE THE FOLLOWING COMMAND STRING:
;
;     PMA XTR,XTR/-SP=LB:[1,5]NETLIB/ML,XTR
;
; TO TASK BUILD USING THE TOOL KIT, USE THE FOLLOWING COMMAND STRING:
;
;     XTR/PR:0,XTR/-SP=XTR,LB:[1,5]NETLIB/LB:GCL
;     /
;     STACK=30
;     UNITS=3
;     ASG=TI:1:2:3
;     TASK=...XTR
;     //
;
;
; THE FOLLOWING IS AN EXAMPLE OF THE XTS DIALOG:
```

(continued on next page)

DLX Receive Program (cont.)

```

;
; >XTS
; LINE: CNA-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: CNA-0
;
; WHEN FINISHED WITH XTS/XTR, XTR MUST BE ABORTED. NOTE - YOU MUST BE
; IN THE TOOL KIT/DCL IN ORDER TO ABORT XTR WITHOUT CAUSING YOUR PRO TO
; BUGCHECK WITH A 300/6 ERROR!
;-

.SBTTL LOCAL MACROS

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

.SBTTL MACRO CALLS

.MCALL QIOW$,QIO$,QIOW$$,ALUN$$,EXIT$$,EXST$$,ASTX$$,WTSE$$
.MCALL GCL$,GCLDF$,DLXDF$,EPMDF$

DLXDF$ ;DEFINE DLX FUNCTION CODES AND OVERHEAD
EPMDF$ ;DEFINE ADDITIONAL ETHERNET VALUES

.SBTTL CONSTANTS
;
; RECEIVE BUFFER SIZE
;
; BUFSIZ = 90.
;
; LUN ASSIGNMENTS:
;
; TILUN=1 ;LUN FOR TI
; CHNLUN=2 ;LUN FOR ERROR FREE CHANNEL
; ERLUN=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

```

(continued on next page)

H-42 TASK-TO-TASK PROGRAMMING EXAMPLES

DLX Receive Program (cont.)

```

;
; 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,XMICHR,XMILEN>

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

;
; SET CHARACTERISTICS BUFFER
;
SETCHR: .WORD   CC.DST                      ;SETTING UP PROTOCOL TYPE/ADDRESS
        .WORD   10.                        ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD   0                          ;RESERVED FIELD
        .WORD   0                          ;STATUS WORD
        .WORD   1                          ;DEFINE PROTOCOL TYPE
        .WORD   LF$PAD                      ;PAD MESSAGES TO MINIMUM LENGTH
        .BYTE   252                        ;DEFINE XTS ADDRESS (DECNET ADDRESS)
        .BYTE   0
        .BYTE   4
        .BYTE   0
        .BYTE   232
        .BYTE   20

SETLEN=.-SETCHR                          ;LENGTH OF SET CHAR BUFFER.

;
; PROTOCOL/ADDRESS BUFFER FOR TRANSMITTING
;
XMICHR: .WORD   CC.ADR                      ;THIS PART DEFINES AN ADDRESS
        .WORD   6                          ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD   0                          ;RESERVED FIELD
        .WORD   0                          ;STATUS WORD
        .BYTE   252                        ;DEFINE XTS ADDRESS (DECNET ADDRESS)
        .BYTE   0
        .BYTE   4
        .BYTE   0
        .BYTE   232
        .BYTE   20
        .WORD   CC.PRO                      ;THIS PART DEFINES A PROTOCOL TYPE
        .WORD   2                          ;LENGTH OF BUFFER PAST STATUS WORD
        .WORD   0                          ;RESERVED WORD
        .WORD   0                          ;STATUS WORD
        .WORD   1                          ;PROTOCOL TYPE = 1
XMILEN=.-XMICHR                          ;LENGTH OF WHOLE XMICHR BUFFER

```

(continued on next page)

DLX Receive Program (cont.)

```

;
; 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

;
; CHANNEL RECEIVE BUFFERS
;
R1BUF:  .BLKB  BUFSIZ                ;FIRST BUFFER DESCRIPTOR
R2BUF:  .BLKB  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
SETERR: .ASCII  /Error Defining Protocol-Address -- /
BUFSET: .BLKB  7
XMTErr: .ASCII  /Error transmitting data -- /
BUFXTM: .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)

DLX Receive Program (cont.)

```

; 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$$  #IO.XOP,#CHNLUN,#CHNEFN,,#CHNSB,,<R4,R5>
    BCS     15$                      ; IF CS, ERROR
    MOVB    CHNSB,R1                 ; SUCCESSFUL ?
    BPL     16$                      ; IF PL, YES
    MOV     #BUFOPN,R0               ; ELSE, GET BUFFER ADDRESS
    CLR     R2                      ; ZERO SUPPRESSION
    CALL    $CBOMG                   ; CONVERT NUMBER
    CLRB    (R0)                    ; MAKE STRING ASCIZ
15$: EPRINT OPNERR                   ; OPEN ERROR
    BR      99$
;
; DEFINE THE ETHERNET ADDRESS AND PROTOCOL TYPE OF THE XTS PROGRAM
;
16$: QIOW$$  #IO.XSC,#CHNLUN,#CHNEFN,,#CHNSB,,<#SETCHR,#SETLEN>
    BCS     17$                      ; IF CS, ERROR
    MOVB    CHNSB,R1                 ; SUCCESSFUL ?
    BPL     20$                      ; IF PL, YES
    MOV     #BUFSET,R0               ; ELSE, GET BUFFER ADDRESS
    CLR     R2                      ; ZERO SUPPRESSION
    CALL    $CBOMG                   ; CONVERT NUMBER
    CLRB    (R0)                    ; MAKE STRING ASCIZ
17$: EPRINT SETERR                   ; OPEN ERROR
    BR      99$
;
; HANG AN ASYNCHRONOUS READ ON LINE
;
20$: CALL    HNGRC1
    BCS     99$                      ;IF CS, ERROR

    CALL    HNGRC2
    BCS     99$                      ; IF CS, ERROR
;
; THE REST IS AST DRIVEN. MAKE BELEIVE WE ARE WAITING FOR SOMETHING !!
;
    WTSE$$  #DONE                    ;WAIT FOR COMPLETION (NEVER HAPPENS!)
;
99$: DIR$    #CLOSE                   ;CLOSE DOWN THE LINE
    EXIT$$  ;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)

DLX Receive Program (cont.)

```

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:

       MOV      R1,(SP)   ; SAVE R1
       MOV      TEMP,R1  ; R1 -> IOSB
       TSTB     (R1)      ; SUCCESSFUL 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

.SBTTL  $EPRINT -- PRINT ERROR MESSAGE
;+
; **-$EPRINT-PRINT ERROR MESSAGE
;
; PRINTS THE SPECIFIED ERROR MESSAGE PREFIXED BY "XTR -- ".
; SETS THE EXIT-STATUS AS "EX$ERR".
;

```

(continued on next page)

DLX Receive Program (cont.)

```

; 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

;   RECEIVE HUNG ON LINE
;
;-
      .ENABL  LSB
HNGRC1: DIR$    #REC1            ;HANG READ
        BCS    10$              ;IF CS, ERROR
        BR     20$              ;AND CONTINUE IN COMMON CODE
HNGRC2: DIR$    #REC2            ;HANG READ
        BCC    20$              ; IF CC RETURN
10$: EPRINT RECERR              ;RECEIVE ERROR
      SEC                      ;INDICATE FAILURE
20$: RETURN
      .DSABL  LSB
-
      .SBTTL  XMIT - TRANSMIT DATA OVER LINE
;+
;   **XMIT - TRANSMIT DATA OVER LINE
;
; INPUTS:
;   NONE.
;
; OUTPUTS:
;   DATA TRANSMITTED
;
;-

XMIT:  MOV     R1,-(SP)          ;SAVE R1
        DIR$    #XMT            ;TRANSMIT DATA
        BCS     10$             ;IF CS, ERROR
        MOVB    CHNSB,R1        ;SUCCESSFUL ?
        BPL     20$             ;IF PL, YES
        MOV     #BUFXTM,R0      ;ELSE, GET BUFFER ADDRESS
        CLR     R2              ;ZERO SUPPRESSION
        CALL    $CBOMG          ;CONVERT NUMBER
        CLRB    (R0)            ;MAKE STRING ASCIZ

```

(continued on next page)

DLX Receive Program (cont.)

```

10$:    EPRINT  XMERR                      ;TRANSMIT ERROR
        SEC                      ;INDICATE FAILURE
20$:    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

```


INDEX

- Aborting a logical link
 - see ABT\$, ABTNT
- ABT\$, 3-12, A-9
- ABTNT, 2-10, A-9
- ACC\$, 3-14, A-5
- Accepting connect requests
 - see ACC\$, ACCNT
- Access control, 2-3, 2-19, 2-22, 2-25, 3-20, 3-21, 3-35
 - general discussion, A-3
 - MACRO-11 tasks, 3-8
 - using CONB\$\$, 3-8
- Accessing a network
 - see OPN\$, OPNNT
- ACCNT, 2-13, A-5
- ADB
 - see Application Diskette Builder
- Addressing modes (Ethernet)
 - multicast, 4-2, 4-6, 4-9, 4-10
 - physical, 4-2
 - setting address pairs, 4-4, 4-8
- Alias, 2-3, 2-16
- Application Diskette Builder, 1-2
- Application installation file, 1-3
 - format conventions, 1-4
 - sample, 1-5
- Assigning a LUN, 2-2
 - using .MBXLU for MACRO-11 tasks, 2-2
- AST
 - see asynchronous system trap
- Asynchronous system trap
 - and the WAIT option [W], 3-6
 - MACRO-11 tasks, 3-6
- BACC, 2-16
- BFMT0, 2-20
- BFMT1, 2-23
- Buffer
 - determining segment size, 2-34, 3-25, A-7
- Buffer size
 - see GLN\$, GLNNT
- BUILD type macro, 3-1
 - format, 3-3, 3-5
 - general description, 3-2
- Closing the network connection
 - see CLS\$, CLSNT
- CLS\$, 3-16
- CLSNT, 2-2, 2-26
 - CLSNTW (wait version), 2-2
- Combined installation file
 - contents, 1-7
 - format conventions, 1-7
 - sample, 1-7
- Completion status
 - see I/O status block
- CON\$, 3-17, A-4
- CONB\$\$, 3-20
- Connect block
 - see BACC, BFMT0, BFMT1, CONB\$\$
 - and connect requests, 3-14
 - building with DECnet calls, A-2
 - contents, A-2
 - retrieving incoming data, A-5
 - size, A-4
- Connect block offsets
 - for BACC, 2-19
 - for BFMT0, 2-22
 - for BFMT1, 2-25
 - for CONB\$\$, 3-21
 - for GND\$, 3-34
- Connect requests, A-2
 - accepting, A-5
 - rejecting, A-5
- CONNT, 2-28, A-4
- Copying files
 - from a VMS node to a Professional 350, 1-12
 - using P/OS DCL command, 1-12
- Creating a logical link
 - see CON\$, CONNT
- DAPRES resident library, 5-2
 - for remote file access, 5-2
 - overlay descriptor file, 5-2
- Data
 - interrupt messages, A-8
 - normal messages, A-7
 - optional messages, 3-35, A-3
 - receiving, A-7
 - retrieving messages, 2-37, 3-28
 - sending, A-7
 - unsolicited messages, A-7
 - verifying message reception, A-7
- DCL, 5-3
- DECNA, 4-2
- DECnet
 - defined, A-1
 - message types, 2-37

I-2 INDEX

- Destination descriptor, 2-19, 2-22, 2-25, 3-20, 3-21, 3-34, A-2
- Digital Command Language
 - see DCL
- DIR\$ directive, 3-2, 3-5
- Direct line access controller
 - see DLX
- Disconnect and abort messages
 - see GND\$, GNDNT
 - retrieving, A-9
- Disconnecting a logical link
 - see DSC\$, DSCNT, A-9
- DLX, 4-1
 - error codes, 4-9, 4-12
 - Ethernet calls summary, 4-3
 - NX: device, 4-1, 4-4
 - timeout periods, 4-5
- DLX QIOs, 4-1, 4-3 to 4-16
 - software standards, 4-3
- DLXDF\$ macro, 4-3
- DSC\$, 3-23, A-9
- DSCNT, 2-31, A-9
- End network task operations, 3-16
- Error conditions
 - duplicate object names, 1-13
 - duplicate object type numbers, 1-13
 - object description file error, 1-13
- Error/completion codes
 - for high level languages, see Appendix E
 - for MACRO-11, see Appendix F
 - categories, 2-3
 - for GND\$, 3-28
- Establishing a network task, A-2
 - MACRO-11 tasks, 3-8
- Ethernet
 - auxiliary characteristics buffer, 4-10, 4-13, 4-14
 - characteristics buffer format, 4-7
 - closing the channel (IO.XCL), 4-16
 - destination address, 4-2
 - opening the channel (IO.XOP), 4-4
 - padding messages, 4-3
 - protocol type, 4-2, 4-4, 4-6, 4-8, 4-10, 4-13
 - receiving a message (IO.XRC), 4-13
 - routing methods, 4-2
 - set characteristics (IO.XSC), 4-6
 - setting destination address, 4-11
 - setting protocol type, 4-11
 - special considerations, 4-2
 - transmitting a message (IO.XTM), 4-10
- Event flag
 - and the WAIT option [W], 3-6
 - NOFLOW, 3-9, 3-18
 - NT.LON, 3-30, 3-32
 - NT.TYP, 3-32
- EXECUTE type macro, 3-1
 - format, 3-3, 3-5
 - general description, 3-3
- FAL, 5-1
- File access
 - differences between local and remote, 5-2
- File Access Listener
 - see FAL
- Flow control, 3-23
 - and DLX, 4-1
 - DECnet response, 3-9
 - high level language tasks, 2-3
 - MACRO-11 tasks, 3-8
 - NOFLOW option, 3-9
 - REC\$, 3-39
 - source and target tasks, 3-9
- Full duplex transmission, A-7
- GLN\$, 3-25, A-7
- GLNDT, A-7
- GLNNT, 2-34
- GND\$, 3-14, 3-28, A-5, A-8, A-9
- GNDNT, 2-37, A-5, A-8, A-9
- High level language communication calls
 - summary, 2-7
- I/O status block
 - contents of second word using GNDNT, 2-38
 - general discussion, A-8
 - MACRO-11 values, 3-7
 - values for GND\$, 3-30
 - word values, 2-3, A-9
- INS file
 - different types, 1-3, 1-7
 - format conventions, 1-4
 - installing, 1-4
 - removing, 1-4
- Interrupt messages
 - function, A-8
 - maximum number, A-8
 - retrieving, A-8
 - sending, A-8
 - size, A-8
 - using XMI\$, 3-47
- IO.XCL, 4-16

- IO.XOP, 4-4
- IO.XRC, 4-13
- IO.XSC, 4-6
- IO.XTM, 4-10
- Libraries
 - NETDEF.PAS, 1-3
 - NETLIB.MLB, 1-3, 3-1, 4-3
 - NETSUB.OLB, 1-3, 2-1
- Link recovery period, 2-44, 3-38
- Local node information
 - see GLN\$, GLNDT
- Local node name, 2-34, 3-25
- Logical link
 - aborting, 3-12, A-9
 - accepting, A-5
 - closing the network connection, A-10
 - creating, 3-17, A-4
 - disconnecting, 3-23, A-9
 - general description, A-4
 - rejecting, A-5
 - sending data, A-7
 - terminating activity on, A-9
- Logical unit number
 - see LUN
- lrp*
 - see link recovery period
- LUN
 - assigning, A-4
 - assigning with .MBXLU macro, 3-8
 - specifying, A-4
 - used in DECnet calls, A-4
- Macro failures, 3-5
- MACRO-11
 - error/completion code format, 3-7
 - example call formats, 3-5
 - using AST routines, 3-6
 - using event flags, 3-6
 - using the WAIT option [W], 3-6
- MACRO-11 communication calls
 - summary, 3-10
- MACRO-11 connect block offsets
 - see Appendix D
- Named object
 - defined, A-2
 - for a high level language task, A-2
 - for a MACRO-11 task, A-2
- Network
 - accessing, A-5
 - efficient message transmission, 2-34, 3-25, A-7
- Network data queue, 3-16, A-2
 - see GND\$, GNDNT
 - closing status of pending DECnet calls, A-10
 - creating, A-5
 - deleting, A-10
 - ways to retrieve messages, 2-37, 3-28
- Network disconnects and rejects
 - see Appendix B
- Network object
 - defined, A-2
- Node number, 2-34, 3-25
- NOFLOW option, 3-18
- NT.LON
 - and GND\$ I/O status blocks, 3-30
 - function, 3-32
- NT.TYP
 - function, 3-32
- Numbered object
 - defined, A-3
 - for a high level language task, A-3
 - for a MACRO-11 task, A-3
- Object description file
 - /COPIES switch, 1-9
 - file specifications, 1-8
 - OBJECT command, 1-8
 - requirements, 1-8
 - RUN/INSTALL command, 1-10
 - RUN/REMOVE command, 1-10
 - samples, 1-11
 - /VERIFICATION switch, 1-10
- Object installation file, 1-3
 - contents, 1-5
 - EXECUTE directive, 1-6
 - format conventions, 1-5
 - installing an object, 1-6
 - object description file, 1-6
 - removing an object, 1-6
 - sample, 1-6
- Object task
 - debugging on a Professional 350, 1-12
 - setting verification level, 1-10, 3-30
- Object type codes
 - see Appendix C
- OPN\$, 3-8, 3-16, 3-37, A-2, A-5
 - link recovery period, 3-38
- OPNNT, 2-2, 2-43, A-2, A-5
 - link recovery period, 2-44
 - OPNNTW (wait version), 2-2
- Optional arguments, 2-5
 - using paired arguments in DECnet calls, 2-5
 - using single arguments in DECnet calls, 2-6
- Optional data message, 3-35, A-3
 - part of other messages, A-9
 - size, A-9

I-4 INDEX

- P/OS, 1-5
- PAB, 1-3, 2-1
 - cluster library, 5-2
 - using libraries, 1-3
- PMA, 3-1
- PRO/DECnet, 1-1
 - and CLS\$, 3-16
 - and DSC\$, 3-23
 - application development cycle, 1-2
 - applications defined, 1-3
 - building tasks, 1-3
 - "combined" applications defined, 1-4
 - object tasks defined, 1-3
 - software features, 1-2
 - using a trace routine, 1-12
 - using the Ethernet device (DECNA), 4-4
- PRO/Tool Kit, 1-2, 1-12
- Professional 350 computer, 1-1
- Professional Application Builder
 - see PAB
- Professional Host Tool Kit, 1-2
- Professional Macro Assembler
 - see PMA
- Professional Operating System
 - see P/OS
- Queue
 - see Network data queue
- REC\$, 3-39, A-7
- Receiving data
 - see REC\$, RECNT
- RECENT, 2-31, 2-47, A-7
- REJ\$, 3-41, A-5
- Rejecting connect requests
 - see REJ\$, REJNT
- REJNT, 2-50, A-5
- Remote file access, 5-1
 - error/completion codes, see Appendix G
 - alias, 2-3
 - pool considerations, 5-4
 - target system conventions, 1-12
 - using PRO/DECnet for, 1-12, 5-2
- Remote node
 - specification formats, 5-3
- Retrieving connect blocks
 - see GND\$, GNDNT
- Retrieving interrupt messages
 - see GND\$, GNDNT, A-8
- RMS-11, 5-1
- Sending data
 - see SND\$, SNDNT
- Sending interrupt messages
 - see XMI\$, XMINT
- SND\$, 3-23, 3-43, A-7
- SNDNT, 2-31, 2-53, A-7
- Software compatibility, 1-12
- Source descriptor, 3-34, A-3
- Source task, A-8
 - defined, A-4
- SPA\$, 3-28, 3-45
 - programming example, 3-46
- STACK type macro, 3-1
 - format, 3-4, 3-5
 - general description, 3-4
- Symbolic offsets
 - see Connect block offsets
- Target node name, 3-20
- Target system
 - performing access verification, A-3
- Target task, A-8
 - defined, A-4
- Task building, 2-1
 - using BUILD and EXECUTE type macros, 3-2
- Task-to-task communication
 - capabilities, A-1
 - DECnet calls summary, A-11 to A-12
 - programming examples, H-1 to H-47
 - using DLX, 4-1
- Trace routine
 - in a PRO/DECnet program, 1-12
- WAIT option [W], 3-6
 - event flag, 3-6
- WAITNT, 2-56
- XMI\$, 3-47, A-8
- XMINT, 2-58, A-8

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____

or
Country

--- Do Not Tear - Fold Here and Tape ---

digital



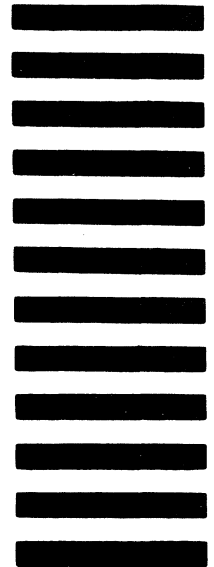
No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754



--- Do Not Tear - Fold Here and Tape ---