DECnet–VAX
User's Guide

Order No. AA-H802B-TE

May 1982

This manual describes user-level functions including remote file access and task-to-task communication using DECnet-VAX software running on a VAX/VMS operating system.

REVISION/UPDATE INFORMATION: This revised document supersedes the *DECnet-VAX User's Guide* (Order No. AA-H802A-TE).

SOFTWARE VERSION: VAX/VMS V3.0

digital equipment corporation · maynard, massachusetts

First Printing, March 1981
Revised, May 1982

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

ZK2183

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

CONTENTS

Page

PREFACE ix

CHAPTER 1 DECNET-VAX OVERVIEW

1.1 VAX/VMS USER INTERFACE TO THE NETWORK 1-1
1.1.1 DECnet-VAX Network Operations 1-1
1.1.2 A Network Topology 1-4

CHAPTER 2 ACCESSING THE NETWORK

2.1 FILE AND TASK SPECIFICATIONS 2-1
2.2 ACCESS CONTROL 2-5
2.3 USING LOGICAL NAMES 2-9
2.3.1 Iterative Translation 2-10
2.3.2 Names Prefixed by an Underscore Character . . 2-11
2.4 NETWORK COMMAND TERMINALS 2-11

CHAPTER 3 REMOTE FILE ACCESS USING DCL

3.1 ACCESSING THE NETWORK USING DCL COMMANDS 3-2
3.2 LOGICAL NAME COMMANDS 3-4
3.2.1 ASSIGN, DEASSIGN, and DEFINE 3-4
3.2.2 SHOW LOGICAL and SHOW TRANSLATION 3-5
3.3 COMMANDS FOR FILE HANDLING 3-5
3.3.1 ANALYZE/RMS_FILE 3-7
3.3.2 APPEND and COPY 3-8
3.3.3 BACKUP 3-8
3.3.4 CONVERT 3-9
3.3.5 CREATE 3-10
3.3.6 DELETE and PURGE 3-10
3.3.7 DIFFERENCES 3-11
3.3.8 DIRECTORY 3-11
3.3.9 DUMP/RECORDS 3-12
3.3.10 PRINT/REMOTE 3-12
3.3.11 SEARCH 3-13
3.3.12 SORT and MERGE 3-13
3.3.13 SUBMIT/REMOTE 3-14
3.3.14 TYPE 3-15
3.4 LEXICAL FUNCTIONS 3-15
3.4.1 F$FILE_ATTRIBUTES 3-15
3.4.2 F$PARSE 3-16
3.4.3 F$SEARCH 3-16
3.5 COMMANDS FOR ACCESSING RECORDS 3-16
3.5.1 OPEN and CLOSE 3-17
3.5.2 READ and WRITE 3-17
3.6 COMMAND PROCEDURE EXAMPLES 3-18
3.6.1 Command Procedure Using Lexical Functions . . 3-18
3.6.2 Command Procedure Using SYS$NET 3-18

Page

3.7 DISPLAY OF ERROR MESSAGES IN NETWORK ENVIRONMENT 3-19

CHAPTER 4 REMOTE FILE ACCESS USING RMS

4.1 ACCESSING THE NETWORK AT THE RMS LEVEL 4-1
4.2 VAX-11 NETWORK FILE ACCESS RESTRICTIONS 4-2
4.3 VAX-11 RMS NETWORK ERROR REPORTING 4-3
4.4 HIGHER-LEVEL LANGUAGE REMOTE FILE ACCESS 4-7
4.5 MACRO REMOTE FILE ACCESS 4-8
4.5.1 Using VAX-11 RMS Service Calls 4-9
4.5.1.1 File Access Blocks (FABs) and Record Access Blocks (RABs) 4-9
4.5.1.2 $OPEN and $CONNECT 4-9
4.5.1.3 $DISCONNECT and $CLOSE 4-9
4.5.2 VAX-11 RMS Service Call Summary 4-9
4.5.3 VAX-11 RMS Programming Notes and Restrictions 4-11
4.5.3.1 Name Block 4-11
4.5.3.2 File Specification Processing 4-12
4.5.3.3 FOP File Disposition Options on Close . . . 4-12
4.5.3.4 FOP Option for Increasing File Transfer Throughput 4-12
4.5.3.5 File Sharing 4-13
4.5.3.6 Restriction on Access to Files on Magnetic Tape 4-14
4.5.3.7 Task-to-Task Communication 4-14
4.5.4 MACRO Programming Examples 4-14
4.5.4.1 MACRO Remote File Transfer Example 4-14
4.5.4.2 VAX-11 MACRO Remote File Spooling Example 4-17
4.5.4.3 VAX-11 MACRO Remote File Random Access Example 4-18
4.5.4.4 VAX-11 MACRO Remote File Indexed Access Example 4-20

CHAPTER 5 TASK-TO-TASK COMMUNICATION

5.1 TRANSPARENT COMMUNICATION 5-1
5.2 NONTRANSPARENT COMMUNICATION 5-4
5.2.1 Mailboxes and Mailbox Messages 5-5
5.3 INITIATING A LOGICAL LINK CONNECTION 5-7
5.3.1 The Handshaking Sequence 5-7
5.4 COMPLETING THE LOGICAL LINK CONNECTION 5-8
5.4.1 Completing the Connection Transparently 5-8
5.4.2 Completing the Connection Nontransparently . . . 5-9
5.4.3 Command Procedures Used in Task-to-Task Communication 5-9
5.5 EXCHANGING MESSAGES 5-10
5.5.1 Data Messages 5-10
5.5.2 Mailbox Messages 5-10
5.6 TERMINATING THE COMMUNICATION PROCESS 5-11

CHAPTER 6 TRANSPARENT TASK-TO-TASK COMMUNICATION USING SYSTEM SERVICES

6.1 SYSTEM SERVICE CALLS FOR TRANSPARENT COMMUNICATION 6-1
6.2 REQUESTING A LOGICAL LINK 6-2
6.3 COMPLETING THE LOGICAL LINK CONNECTION 6-2
6.4 EXCHANGING MESSAGES 6-3
6.5 TERMINATING THE LOGICAL LINK 6-3
6.6 STATUS AND ERROR REPORTING 6-3
6.7 SYSTEM SERVICE CALL SUMMARY 6-3
6.7.1 $ASSIGN (I/O Channel Assignment) 6-4

CONTENTS

Page

6.7.2 $QIO (Sending a Message to a Target Task) . . . 6-5
6.7.3 $QIO (Receiving a Message from a Target Task) . 6-6
6.7.4 $DASSGN (Terminating a Logical Link) 6-7
6.8 PROGRAMMING EXAMPLE OF TRANSPARENT COMMUNICATION . 6-8

CHAPTER 7 NONTRANSPARENT TASK-TO-TASK COMMUNICATION USING SYSTEM SERVICES

7.1 SYSTEM SERVICE CALLS FOR NONTRANSPARENT COMMUNICATION 7-1
7.2 ASSIGNING A CHANNEL TO _NET: AND CREATING A MAILBOX 7-2
7.2.1 Mailbox Message Format 7-3
7.3 REQUESTING A LOGICAL LINK CONNECTION 7-4
7.3.1 Network Connect Block 7-4
7.4 COMPLETING THE LOGICAL LINK 7-5
7.4.1 Receiving a Connection Request 7-6
7.4.1.1 Receiving Single Connection Requests . . . 7-6
7.4.1.2 Receiving Multiple Connection Requests (Network Tasks) 7-7
7.4.2 Accepting or Rejecting a Connection Request . . 7-7
7.5 EXCHANGING MESSAGES 7-7
7.5.1 Interrupt Messages 7-7
7.6 DISCONNECTING OR ABORTING THE LOGICAL LINK 7-8
7.6.1 Synchronously Disconnecting a Logical Link . . . 7-8
7.6.2 Aborting the Logical Link ($QIO Function) . . . 7-8
7.6.3 Terminating the Logical Link ($DASSGN Function) 7-9
7.7 STATUS AND ERROR REPORTING 7-9
7.8 SYSTEM SERVICES CALL SUMMARY 7-9
7.8.1 $ASSIGN (I/O Channel Assignment) 7-9
7.8.2 $QIO (Requesting a Logical Link Connection) . 7-10
7.8.3 $QIO (Accepting a Logical Link Connection Request) 7-12
7.8.4 $QIO (Rejecting a Logical Link Connection Request) 7-13
7.8.5 $QIO (Sending a Message to a Target Task) . . 7-14
7.8.6 $QIO (Receiving a Message from a Target Task) 7-14
7.8.7 $QIO (Sending an Interrupt Message to a Target Task) 7-14
7.8.8 $QIO (Synchronously Disconnecting a Logical Link) 7-15
7.8.9 $QIO (Aborting a Logical Link) 7-16
7.8.10 $QIO (Declaring a Network Name or Object Number) 7-17
7.8.11 $DASSGN (Terminating a Logical Link) 7-18
7.9 PROGRAMMING EXAMPLE FOR NONTRANSPARENT COMMUNICATION 7-18

APPENDIX A OBJECT TYPE CODE VALUES

APPENDIX B VAX-11 RMS CONTROL BLOCK USE

APPENDIX C SUMMARY OF NETWORK SYSTEM SERVICE ERROR MESSAGES

APPENDIX D MAILBOX MESSAGE TYPES

GLOSSARY

Page

INDEX

EXAMPLES

EXAMPLE 4-1 FORTRAN Remote File Access Program 4-8
4-2 RMS File Transfer Program 4-15
4-3 RMS Spool File Program 4-17
4-4 RMS Random Access Program 4-18
4-5 RMS Indexed File Program 4-20
5-1 FORTRAN Task-to-Task Communication 5-2
6-1 Transparent Task-to-Task Communication Using System Services 6-8
7-1 Nontransparent Task-to-Task Communication Using System Services 7-19

FIGURES

FIGURE 1-1 Network Access Levels and DECnet-VAX User Interface 1-3
1-2 Network Topology and Related Functions 1-4
1-3 Operational Capabilities of DECnet-VAX 1-5
2-1 Remote File Access Using Access Control String Information 2-6
2-2 Remote File Access Using Default Access Control Information 2-7
2-3 Outbound and Inbound Connect Flowcharts 2-8
4-1 Remote File Access (DCL and RMS) 4-2
5-1 Mailbox Messages 5-6
7-1 Mailbox Message Format 7-3
7-2 Network Control Block Format 7-5

TABLES

TABLE 1-1 Network Access Levels 1-2
3-1 DCL Command Summary 3-2
4-1 VAX-11 RMS File and Record Characteristics for Network Operations 4-3
4-2 VAX-11 RMS Access Modes for Network Operations . . 4-3
4-3 Network-Specific RMS Completion Codes 4-4
4-4 VAX-11 RMS Service Calls for Run-time Remote File Access 4-10
6-1 Transparent Task-to-Task Communication System Service Summary 6-1
7-1 Summary of System Service Calls for Nontransparent Task-to-Task Communication 7-2
A-1 Object Type Codes A-1
B-1 FAB (File Access Block) B-2
B-2 RAB (Record Access Block) B-5
B-3 NAM (Name Block) B-7
B-4 XABALL (Allocation Control XAB) B-9
B-5 XABDAT (Date and Time XAB) B-10
B-6 XABFHC (File Header Characteristics XAB) B-10
B-7 XABKEY (Key Definition XAB) B-11
B-8 XABPRO (File Protection XAB) B-12
B-9 XABRDT (Revision Date and Time XAB) B-13
B-10 XABSUM (Summary XAB) B-13

CONTENTS

Page

C-1 System Services Error Messages Summary C-1
D-1 Mailbox Message Summary D-1

PREFACE

MANUAL OBJECTIVES

The DECnet-VAX User's Guide describes VAX/VMS network operations such as remote file access and task-to-task communication. The manual is divided according to the types of operations you can perform and the level at which you access the network. In particular, this manual describes the DIGITAL Command Language (DCL) commands you can use to manipulate remote files and for network command terminal use. It also describes all the information necessary to program remote file access and task-to-task communication applications.

INTENDED AUDIENCE

This manual provides information needed by VAX/VMS users who want to perform network operations. Interactive and batch users should be familiar with DCL, which the VAX/VMS Command Language User's Guide describes in detail. Programmers should use the information in this manual as a supplement to the reference manuals and user guides provided for higher-level languages, and to the manuals that explain VAX-11 Record Management Services (RMS) and VAX/VMS System Services.

STRUCTURE OF THIS DOCUMENT

The DECnet-VAX User's Guide consists of seven chapters, four appendixes, and a glossary, which are described below:

- Chapter 1 briefly describes the ways you can access the network and the types of network operations you can perform. A general network topology, which serves as a common reference for all examples, is also included.

- Chapter 2 describes general concepts that pertain to accessing the network at the command level and through user programs.

- Chapter 3 presents the DCL commands that you can use for network operations.

- Chapter 4 presents the VAX-11 RMS calls and procedures that you can use to access files on remote nodes.

- Chapter 5 describes the concepts central to DECnet-VAX task-to-task communication and the use of higher-level languages for such communication.

- Chapter 6 summarizes the procedures for transparent task-to-task communication using system services.

- Chapter 7 summarizes the procedures for nontransparent task-to-task communication using system services.
- Appendix A provides a table of object type code values and their descriptions.
- Appendix B provides a complete table of RMS control block fields and any related qualifications governing their use for network operations.
- Appendix C summarizes system service error messages associated with network-related functions.
- Appendix D summarizes mailbox messages and their meanings for nontransparent communication.
- The Glossary defines terms used in this manual.

ASSOCIATED DOCUMENTS

For information concerning DECnet-VAX system management, refer to the _DECnet-VAX System Manager's Guide_ and the _DECnet-VAX Network Installation_ manual. The _DECnet-VAX Cross-System Notes_ describe the use of DECnet-VAX in a heterogeneous network environment. The _VAX/VMS Release Notes_ describe any constraints on using DECnet-VAX with the current version of VAX/VMS. The _VAX-11 Information Directory and Index_ provides information on the entire VAX/VMS document set. The directory briefly describes all the documents in the set and explains the intended audience for each one. For general background information about the VAX/VMS system, refer to the _VAX/VMS Primer_ and the _VAX/VMS Summary Description and Glossary_. Finally, the _Introduction to DECnet_ manual serves as a companion to the DECnet-VAX documentation set. This manual provides a general overview of DECnet software from a perspective independent of individual DECnet implementations.

Because the user guides for the higher-level languages available under VAX/VMS contain information relating to remote file access and task-to-task communication, you should refer to them for applications in these areas.

The following functional specifications define DIGITAL Network Architecture (DNA) protocols to which all implementations of DECnet adhere:

DECnet DIGITAL Network Architecture General Description

DIGITAL Data Communications Message Protocol Functional Specification

Network Services Protocol Functional Specification

Maintenance Operation Protocol Functional Specification

Data Access Protocol Functional Specification

Transport Functional Specification

DNA Session Control Functional Specification

Network Management Functional Specification

Graphic Conventions

character (or chars)	The term characters refers to the set of alphanumerics that includes A through Z, 0 through 9, _, and $.

`$ COPY`
`$_FROM: *.*`
`$_TO:   TRNTO::*.*`
	Command examples show all output lines or prompting characters that the system prints or displays in black letters.
	
	This document uses red lettering to indicate all user-entered information and to show user-supplied call instruction parameters.

`$DASSGN_S  chan`	All calls and command example verbs are shown in a call or command line in capital letters, and they must be entered as shown. Arguments are shown in a call as lowercase letters. You substitute the argument shown in the call format with the precise information requested.
`$ASSIGN_S devnam,chan[,acmode]`	Square brackets enclose optional keywords and arguments. Do not include the brackets when entering the command.
"foreign-file-spec-string" "task-spec-string"	Keywords or arguments within braces indicate that you must choose only one of the keywords or arguments. (Do not include the braces when entering the command.)

`$ TYPE BOSTON::TEXT.COM`
.
.
.
(or)
`DELETE file-spec,...`
	The use of ellipses means that not all of the information that the system would display in response to the particular command is shown, or, that not all the information a user would enter is shown.

CHAPTER 1

DECNET-VAX OVERVIEW

DECnet is the collective name for a set of software and hardware products that allow DIGITAL operating systems to participate in a cooperative environment known as a network. DECnet-VAX is the software package that extends the basic capabilities of the VAX/VMS operating system on a VAX-11 computer. Beyond the normal VAX/VMS capabilities, DECnet-VAX provides a layered structure of protocols for network operations. These protocols allow you to use the resources on remote DIGITAL computers, even though these systems may run under an operating system other than VAX/VMS. This manual describes the DECnet-VAX facilities for accessing and using these resources.

This chapter describes the DECnet-VAX user interface to the network in terms of a hypothetical network topology. Of course, each network will be tailored to the individual needs of its users and the resources available. The network topology presented in this chapter serves as a common reference for the user examples presented in this manual and highlights the operational capability of DECnet-VAX within a heterogeneous network environment. In this manual, however, only the DECnet-VAX perspective on network operations is presented. For an overview of DECnet, refer to the Introduction to DECnet manual.

1.1 VAX/VMS USER INTERFACE TO THE NETWORK

The VAX/VMS operating system and DECnet-VAX communications software are integrated to provide a high degree of transparency for user operations. When developing network applications, you can use standard DCL commands, higher-level language I/O statements, VAX-11 RMS service calls, and system service calls to perform network operations. For some applications, however, it is desirable (and sometimes necessary) to have more direct access to network-specific information and operations. For this purpose, DECnet-VAX provides nontransparent communication. The following sections describe general transparent and nontransparent features of DECnet-VAX in terms of the user interface to the network.

1.1.1 DECnet-VAX Network Operations

DECnet-VAX supports a variety of network operations that employ VAX/VMS programming languages. In addition to VAX-11 MACRO, you can use most of the higher-level languages such as VAX-11 FORTRAN, VAX-11 BASIC, VAX-11 BLISS, VAX-11 PASCAL, VAX-11 PL/I, and VAX-11 COBOL to develop networking applications. With any of these languages, you can access remote files and create tasks that exchange information across the network. You can also use DIGITAL Command Language (DCL) commands to create and access remote files and to perform many file management functions on other nodes.

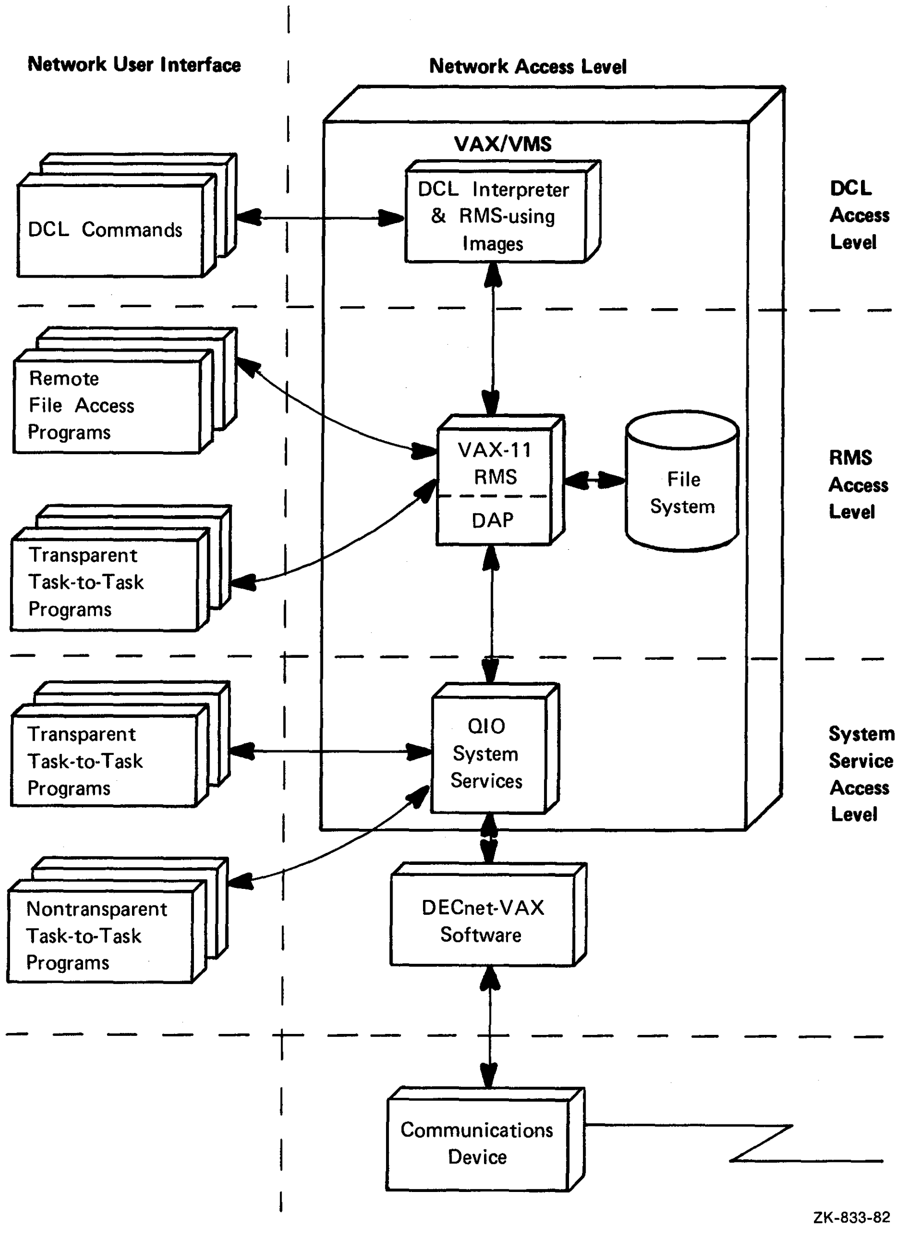
Note that throughout the manual, the term **task** refers to an image running in the context of a process, the term **local** refers to the node at which you are located physically, and the term **remote** refers to any node on the network other than the one at which you are located.

Table 1-1 summarizes the normal use of the programming languages for specific network operations that you can perform with DECnet-VAX.

Table 1-1: Network Access Levels

User Language	Network Operation	Language Calls	Access Level
DCL	Network command terminals Remote file manipulation Task-to-task communication	DCL commands	Transparent network access via DCL
Higher-level languages	Remote file access (files and records) Task-to-task communication	Higher-level language I/O statements	Transparent network access via RMS
MACRO or higher-level languages	Remote file access (files and records) Task-to-task communication	RMS service calls	
MACRO or higher-level languages	Task-to-task communication	System service calls	Transparent and nontransparent network access via QIO

The way you access the network is directly related to the language you use and the network operation you perform. For example, you may want to use standard VAX-11 RMS calls in a VAX-11 MACRO program to access remote files, then use system service calls to communicate between MACRO programs in a task-to-task communication application. Figure 1-1 shows three access levels and the corresponding network operations. The various levels of network access provide a convenient context in which to discuss typical user operations over the network.



Network User Interface
Network Access Level
VAX/VMS
DCL Commands
DCL Interpreter & RMS-using Images
DCL Access Level
Remote File Access Programs
VAX-11 RMS
DAP
File System
RMS Access Level
Transparent Task-to-Task Programs
Transparent Task-to-Task Programs
QIO System Services
System Service Access Level
Nontransparent Task-to-Task Programs
DECnet-VAX Software
Communications Device
ZK-833-82

Figure 1-1: Network Access Levels and DECnet-VAX User Interface

The first two levels of access, DCL and RMS, are entirely transparent to the network user. Because you use standard DCL commands and RMS service calls to access remote files, no DECnet-specific calls are required at these levels of access. You need only specify the remote node on which the file resides in your file specification. Likewise, higher-level language tasks can use a variation of the standard VAX/VMS file specification in conjunction with standard I/O statements to access remote tasks and exchange information; thus, this form of task-to-task communication is also transparent. As with device-independent input/output (I/O) operations, transparent network access allows you to move data across the network with little concern for the way this operation is performed.

The third level of access, system services, provides both a transparent and a nontransparent user interface to the network. Transparent communication at the system service level provides all the basic functions necessary for two tasks to exchange messages over the network. As with the higher-level language I/O interface, these operations are transparent because they do not require DECnet-specific calls. Rather, you use standard system service calls to implement them. Nontransparent communication extends this basic functionality to allow a nontransparent task to receive multiple inbound connections and to use additional network protocol features such as optional user data and interrupt messages. As with device-dependent I/O, nontransparent communication allows you to exploit certain network-specific characteristics to coordinate a more controlled communication environment for exchanging information.

1.1.2 A Network Topology

To highlight the operational capabilities of DECnet-VAX, this section presents a hypothetical network example comprised of various DIGITAL operating systems. Figure 1-2 illustrates a network topology (or geometry) which includes several VAX/VMS systems. No two networks are likely to have the same distribution of resources or the same topology; therefore, this example serves only to illustrate the use of DECnet-VAX functions in a heterogeneous network environment. The examples throughout this manual refer back to the network topology presented here.

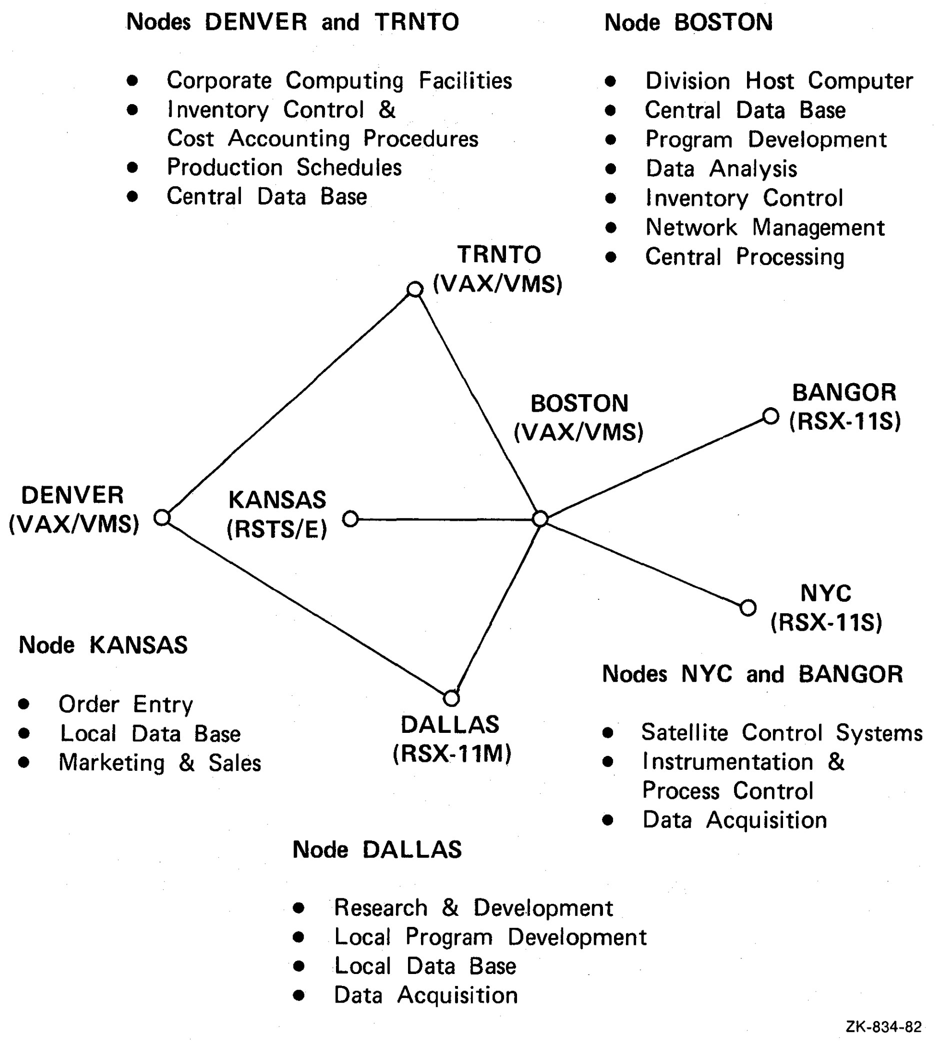


Figure 1-2: Network Topology and Related Functions

The computer network in Figure 1-2 has a centralized VAX/VMS host processor (BOSTON) and remote distributed systems dedicated to particular functions. For the purpose of discussion, this network represents a corporate division that oversees such functions as research and development, process control, and order entry. In this example, the size and functions of the network are simplified.

The VAX/VMS host processor at node BOSTON performs major computation, controls the central data base, and supervises the general operation of the network for the division. The remote computers (nodes) perform various network-related functions in concert with the host processor. The VAX/VMS computers at nodes TRNTO and DENVER represent corporate computing facilities, where the corporation generates reports, production schedules, and inventory and cost accounting procedures. The host computer provides information to these facilities to account for divisional productivity.

Figure 1-3 indicates the type of typical DECnet-VAX operations that enable the host to serve this role. Nodes NYC and BANGOR are satellite computers involved with instrumentation and process control. The host coordinates the manufacturing process via down-line system and task loading and transparent task-to-task communication for data acquisition and retrieval. (Refer to the DECnet-VAX System Manager's Guide for more information on down-line loading.)

Node DALLAS provides computing power sufficient for conducting experiments and collecting data; research and development involving development of local programs replaces the need to use the resources of the host. The host, however, has access to this remote data base, and controls simple procedures that tasks at both NYC and DALLAS must carry out in concert. A nontransparent task running on the host processor controls the exchange of information between the two tasks.

Finally, marketing has direct input to the host in the form of order entry at node KANSAS. Typically, such input might involve direct access to files or programs that update the central data base. The host, in turn, updates the remote data base with inventory information and other sales-related data.

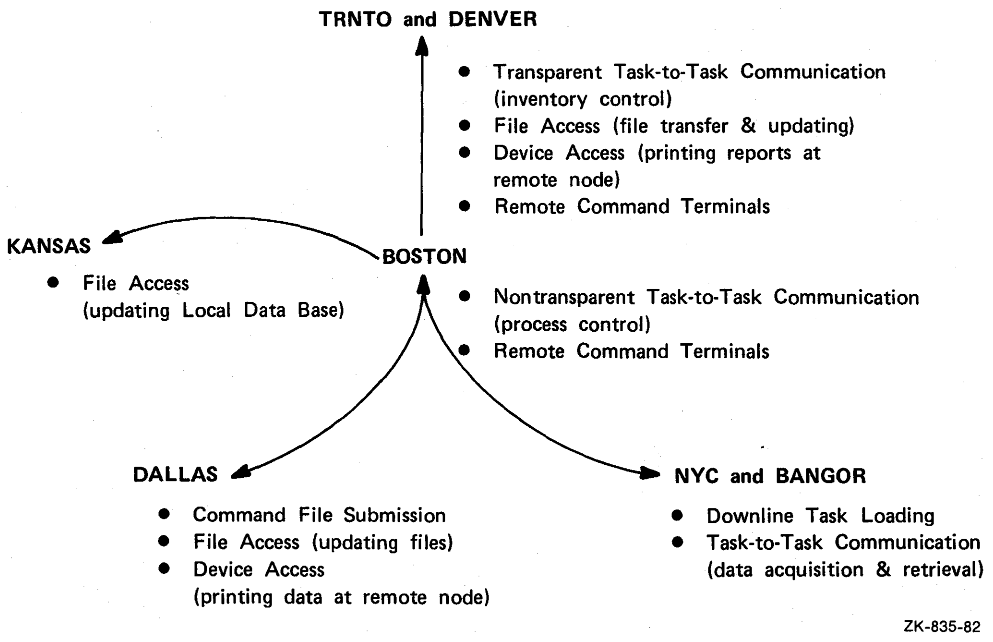

TRNTO and DENVER
• Transparent Task-to-Task Communication (inventory control)
• File Access (file transfer & updating)
• Device Access (printing reports at remote node)
• Remote Command Terminals
KANSAS
• File Access (updating Local Data Base)
BOSTON
• Nontransparent Task-to-Task Communication (process control)
• Remote Command Terminals
DALLAS
• Command File Submission
• File Access (updating files)
• Device Access (printing data at remote node)
NYC and BANGOR
• Downline Task Loading
• Task-to-Task Communication (data acquisition & retrieval)
ZK-835-82

Figure 1-3: Operational Capabilities of DECnet-VAX

Such a network takes advantage of the host's resources efficiently and effectively. The centralized location of the host computer enables it to communicate with all remote nodes performing their individual functions. To control and monitor the various functions of the division, the local VAX/VMS user in interactive or batch mode can use DCL commands to access remote files and devices; the network programmer can develop programs for controlling the manufacturing process and for accessing files on the remote data bases. Remote command terminals allow network users to log in at remote VAX/VMS nodes and perform operations just as they would at the local node. Thus DECnet-VAX provides all the facilities for operating within a network environment while distributing the processing load efficiently among the available network resources.

CHAPTER 2

ACCESSING THE NETWORK

This chapter presents general information that you need to access the network via DECnet-VAX software. This information includes the general format for network file and task specifications, access control parameters, the use of logical names, and the use of network command terminals. The format for file specifications is applicable to file handling operations for both the DCL and the RMS interfaces to the network. The task specification format pertains to task-to-task communication. The information on access control is significant because it defines the way that both local and remote nodes grant access to their system resources. Finally, this chapter discusses the use of logical names, focusing on the flexibility that they provide.

2.1 FILE AND TASK SPECIFICATIONS

DECnet-VAX uses the standard VAX/VMS file specification format for remote file handling and task-to-task communication applications. A node specification string that includes a node name with an optional **access control** string must be present. You use the optional access control string to explicitly specify access information for both files and tasks at the remote node (see Section 2.2). Task-to-task communication requires the use of a **task specification string** enclosed in quotation marks. This string identifies the target task to which you want to connect on a remote node.

Network file specification strings are composed of eight major fields (or elements), which are arranged in the following formats:

$$\text{node-spec::}\left\{\begin{array}{l}\text{device:[directory]filename.type;version}\\ \text{"foreign-file-spec-string"}\\ \text{"task-spec-string"}\end{array}\right\}$$

The punctuation marks and brackets included in these formats are required to separate the fields of the file specification. The double colon (::) after node is treated as a single delimiter. Angle brackets (<>) may be substituted for square brackets ([]) to enclose directory, and a period (.) may be used in place of the semicolon (;) to separate type and version. The total length of a full file specification string may not exceed 252 characters.

VAX-11 RMS converts lowercase characters to uppercase and removes space, tab, and null characters from a file specification, except for characters within quoted strings (such as "access-control-strings", "foreign-file-spec-strings", and "task-spec-strings").

The fields of the full file specification defined below emphasize the fields unique to network processing.

node-spec

identifies the target node and specifies which account on that node to use. If you specify the name (or number) of the local node, the connection is made to the local node by DECnet as if it were a remote node.

The length of a node-spec is 3 to 61 characters, including the required double colon delimiter.

The format of a node-spec is:

$$\left\{\begin{array}{l}\text{nodename}\\ \\ \text{logical-nodename}\end{array}\right\}\text{"access-control-string"::}$$

Each component of the node-spec is defined below:

nodename (1-6 char)	specifies a node in the network. The node name consists of uppercase alphanumeric characters. If the node name is all numeric, it will be interpreted as the node number (where 0 represents the local node). (For compatibility with future versions of DECnet, you should avoid the use of node numbers.) You may prefix the node name with an underscore character (_) to designate that it is not a candidate for logical node name translation.
logical-nodename (1-15 char)	specifies a logical name for a node in the network. Refer to Section 2.3 for rules governing the use of logical node names.
access-control-string (0-42 char)	is an optional quoted character string containing login information that is sent to the remote node. This string designates the remote account under which programs (or tasks) will execute on your behalf or remote files will be accessed to perform the functions that you request. If you omit the access control string, the login information sent to the remote node is the default access control string for that node as specified by the local System Manager (see Section 2.2).

An access control string is expressed in either of the following formats:

"username password"

"username password account"

For a VAX/VMS node, either format is acceptable, but the shorter form is generally used because VAX/VMS ignores the account name subfield when you log in. You should refer to the appropriate DECnet documentation for the access control string format for nodes other than VAX/VMS.

Space and tab characters (or multiples thereof) delimit username, password, and account. Therefore, each substring within the access-control-string may contain any characters except the quote, space, and tab characters.

device, directory, filename, type, and version

are five optional fields that collectively identify the file to be accessed on a remote node. The definitions and syntax rules for these fields are the same for network use as for local file access, including the specification of seven levels of subdirectories and the use of wild card characters where valid. If the syntax of the file specification differs from that of VAX/VMS, use the foreign-file-spec-string format.

If a device or directory is not specified in the file specification string, default values are supplied by the target node, which uses the conventions of its operating system. If the target node is VAX/VMS, all defaults apply as though the user had logged in at the remote node using the access control information implicitly or explicitly supplied.

The *VAX-11 Record Management Services Reference Manual* contains a detailed explanation of the syntax for VAX/VMS file specifications.

foreign-file-spec-string (1-127 char)

is a quoted character string that identifies the file to be accessed on a remote node. The syntax of the file specification must be in the format recognized by the operating system of the remote node.

When you put a file specification between quotation marks, the VAX-11 RMS facility at the local node performs no syntax checking or logical name translation. Rather, the local node passes the file specification intact to the remote node where it is interpreted. Use the quoted string format when the file specification syntax of the remote node differs from that of VAX/VMS. For example, a RSTS file specification may contain a dollar sign ($):

```
$ TYPE KANSAS::"$START.CTL"
```

task-spec-string (2-32 char)

is a quoted string that identifies the remote task to which you attempt the logical link connection. You identify the task by **object type.** An object type is a discrete identifier for either a user task or a known object on a remote node. Object types have two forms:

- Zero (0) plus a name or TASK plus a name (for example, "0=TEST2" or "TASK=TEST2")
- Nonzero without a name (for example, "17=" or "FAL=", where FAL in this case is the name specified for object 17 by the NCP command DEFINE OBJECT)

Nonzero **network objects** (known objects) are intended for generic process addressing over the network. For example, a program written to establish a logical link with the FAL object may send a request by addressing object type 17. User-written tasks are usually addressed as object type 0 plus a name, but they may also be addressed with a nonzero object type alone. (Refer to the Introduction to DECnet manual for a complete discussion of object types and their use.) Appendix A lists the object type numbers reserved for standard DECnet services and those available for user-written programs.

To establish a logical link connection with a remote task addressed as object type 0, use either of the following forms of task specification string, where taskname is one to nine characters in length:

$$\left\{\begin{array}{l}\texttt{"TASK=taskname"}\\ \texttt{"0=taskname"}\end{array}\right\}$$

If the remote node is a VAX/VMS system, the taskname string represents the file name of a DCL command procedure to be executed at the remote node. (If the remote node is not a VAX/VMS system, the maximum length of the task name that it accepts may be different.) The command procedure can complete the logical link itself or it can include a DCL RUN command to execute a program that completes the logical link.

To address the remote task by a nonzero object type, use the following form as the task specification string, where n is an object type number (in the range of 1 to 255) and xyz is an object name equated to a number via the NCP command DEFINE OBJECT:

$$\left\{\begin{array}{l}\texttt{"n="}\\ \texttt{"xyz="}\end{array}\right\}$$

Examples of network file specifications

1. DALLAS::TEST.DAT

   Use this file specification to access the file TEST.DAT on node DALLAS.

2. TRNTO::DMA2:[INVENTORY]TEST.DAT;3
   TRNTO"KC JONES"::DBA1:TEST.DAT;3

   The first of these two file specifications provides no explicit access control information but the second one does.

3. KANSAS::"$START.CTL"
   TOPS20"KC JONES APOLLO"::"A-VERY-LONG-FILE-NAME.TEST.5"

   These examples illustrate the use of quotation marks when the remote node's file specification syntax differs from that of VAX/VMS.

Examples of network task specifications

1. BOSTON::"TASK=TEST2"

   This specification identifies the task TEST2 by using the TASK= form for specifying remote tasks.

2. BOSTON"JOHN SMITH"::"0=TEST2"

   This example is the same as the one above, except that access control information is provided and the alternative 0= form for specifying a task is used.

3. DALLAS::"150="

   This specification identifies the user-defined network object by object type (150).

2.2 ACCESS CONTROL

Access control is the control that a node exercises over inbound logical link connections. The terms **inbound** and **outbound** refer to the direction of the logical link connection request. A node receives and processes inbound requests; it processes and sends outbound requests. This distinction is useful for discussing access control as it relates to VAX/VMS nodes in a network. Refer to appropriate DECnet documentation if the node to which you want to connect is other than VAX/VMS.

When DECnet-VAX software sends an outbound connection request in response to either a remote file access or a task-to-task communication operation, certain access control information may be necessary to connect successfully to the remote node and log in. As in logging in at your local VAX/VMS node, you can supply specific access control information in the form of a user name and password that the remote node recognizes. The remote node processes inbound connection requests containing this information to verify that you are a valid user of the system.

Upon receiving an inbound connection request, DECnet-VAX software at the remote node creates a process and starts the LOGINOUT image, which verifies your access rights by checking the User Authorization File

(UAF). A user account record is set up for you beforehand within this file by the System Manager. Generally, every time you access a network node, your status as a valid user of that node will be verified against the information contained in the UAF. In this instance, access control provides one form of network security. One exception to this form of access control checking is in nontransparent task-to-task communication wherein a task can receive multiple inbound connection requests (see Chapter 7).

There are two ways to supply access control information for network access:

- You can explicitly specify an access control string as part of a file or task specification in the node field.
- You can have the local node forward default or null access control information to the remote node.

In either case, DECnet software sends this information to the remote node which in turn processes the inbound connection request and, if this information is valid, grants access. If you include an access control string as part of a node specification, this information is always sent directly to the remote node. The privileges associated with either the local account under which your process is running or the specified remote account are of no concern to the local DECnet-VAX implementation. Figure 2-1 illustrates the access control processing that takes place for a simple DCL command.

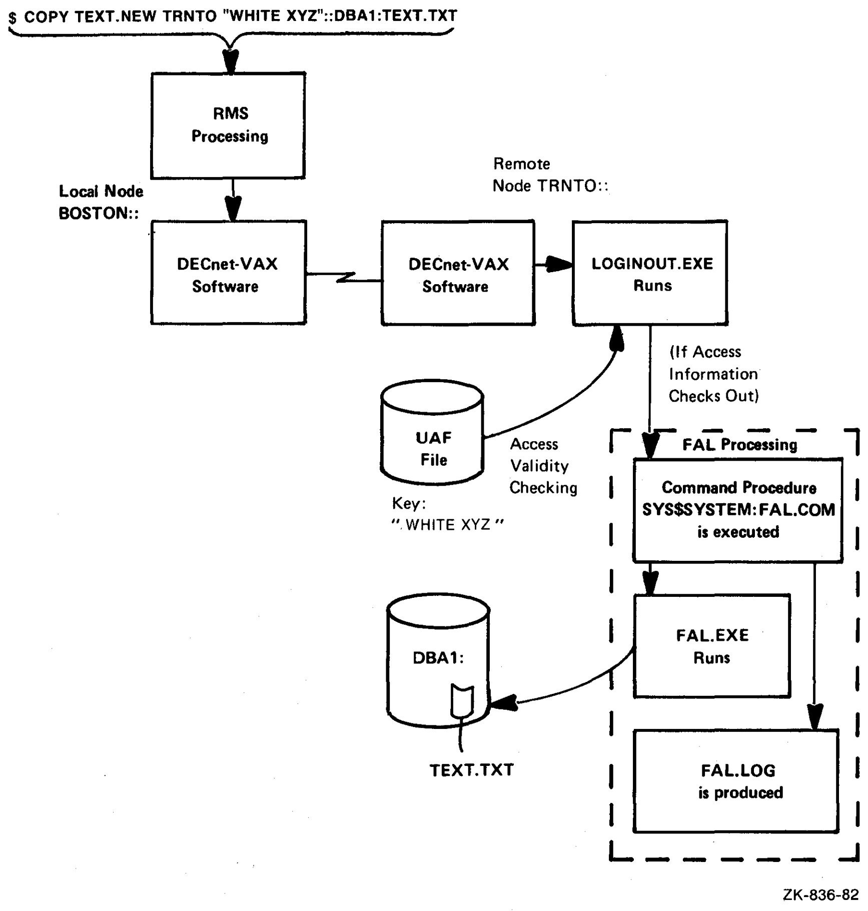


Figure 2-1: Remote File Access Using Access Control String Information

When explicit access control information is not provided in the connection request, DECnet-VAX software uses the remote node name specified in the connection request as a key to locate the appropriate record in the local Configuration Data Base. This record contains default access control information applicable to the remote node. Your System Manager creates this entry when establishing the Configuration Data Base. (Refer to the DECnet-VAX System Manager's Guide for additional information on the Configuration Data Base.) Depending on the privileges required by the object with which you want to connect and those of the user process (see Figure 2-3), one of three possible sets of default access control information is sent to the remote node: default **privileged**, default **nonprivileged**, or null. Because these defaults are node parameters, all privileged operations requested with default access control for a given node run under the same default account. The same is true for nonprivileged operations requested with default access control. Figure 2-2 illustrates the access control processing that takes place for the same DCL command as in the example in Figure 2-1, except that the DCL command does not specify an access control string.

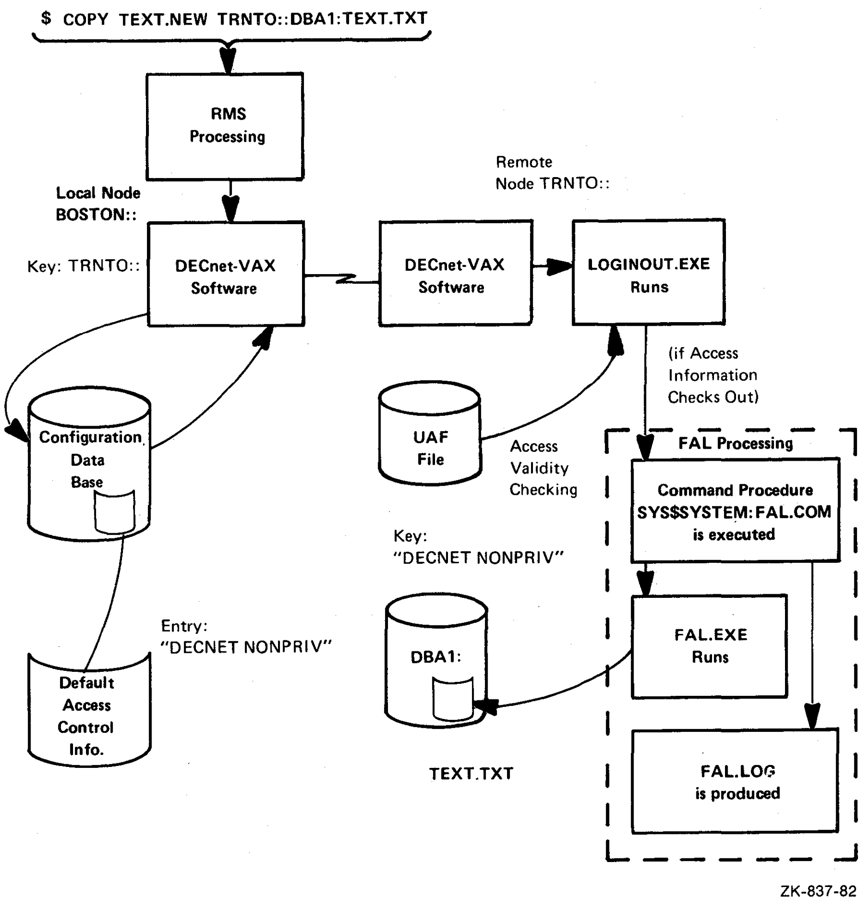


Figure 2-2: Remote File Access Using Default Access Control Information

Note that, in DECnet-VAX usage, **nonprivileged** means no privileges other than TMPMBX and NETMBX. **Privileged** means any privileges in addition to TMPMBX and NETMBX. The context for network-related privileges is the NCP command DEFINE OBJECT. Normally, task-to-task communication and remote file access are nonprivileged operations.

The Configuration Data Base may also define privileges associated with network operations such as task-to-task communication. The System Manager may create object entries with associated privileges. There must be a separate entry for every numbered object that might be requested ("n=").

If, on an outbound connect to an object, you attempt a privileged connection and you do not have sufficient privilege, then you get null access control. On an inbound connection to a nonzero object, any access control information you specify is used. If you do not specify access control information, then one of the following will occur:

- If access control information is associated with the object in the Configuration Data Base, then it is used.
- If no access control information is associated with the object, then the access control for the local nonprivileged network account (if any) is used; otherwise, no information is used.

Access control is not checked for connections to running tasks that have declared names or object numbers (see Section 7.4.1). Figure 2-3 illustrates the process used to determine what access control information (if any) is used for outbound and inbound connections to objects. (Note that in order to use default access control for a privileged account, the process that makes the request must have at least the same local privileges.) Local privileges are completely independent of remote privileges. The DECnet-VAX System Manager's Guide provides a more detailed discussion of access control, network privileges, and the Configuration Data Base.

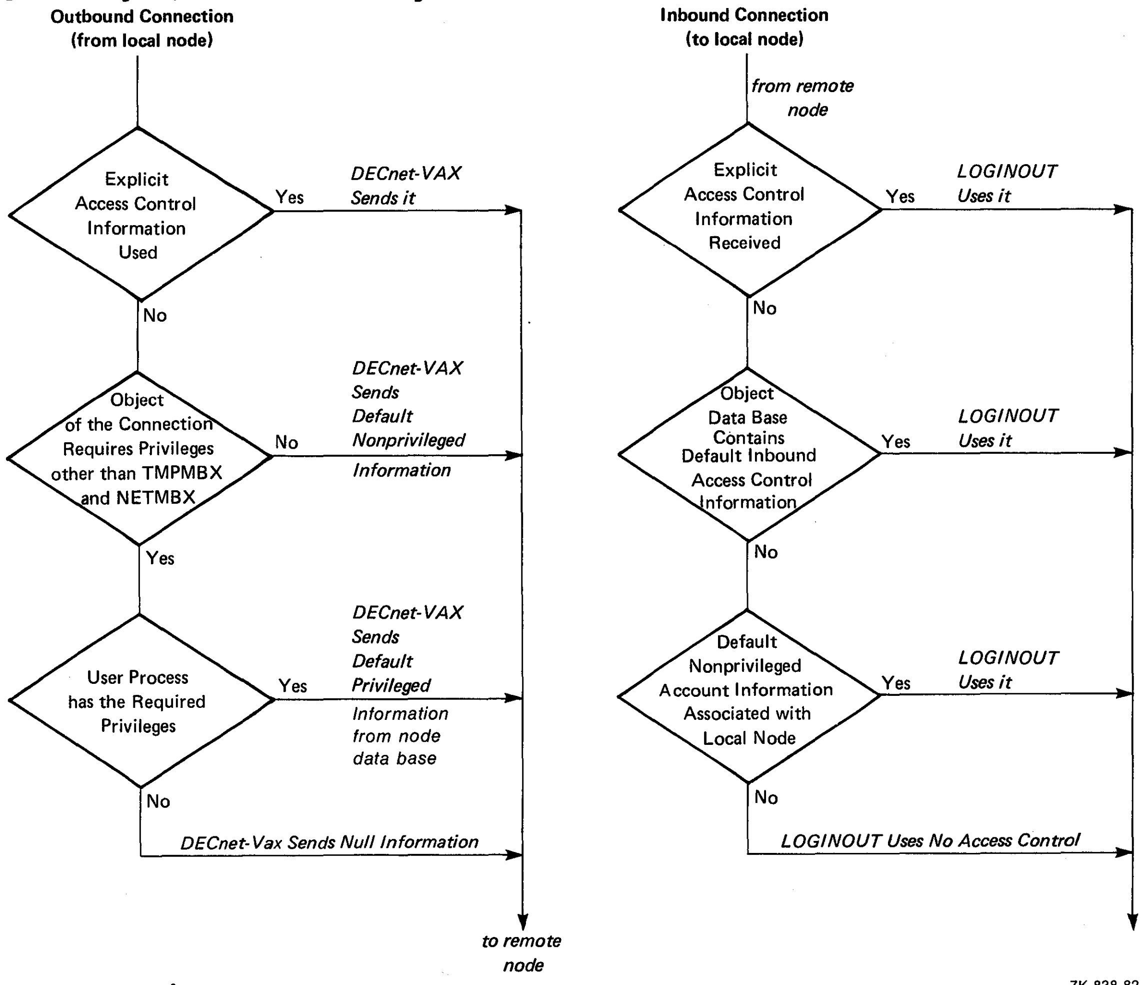

Outbound Connection (from local node)
Explicit Access Control Information Used
Yes
DECnet-VAX Sends it
No
Object of the Connection Requires Privileges other than TMPMBX and NETMBX
No
DECnet-VAX Sends Default Nonprivileged Information
Yes
User Process has the Required Privileges
Yes
DECnet-VAX Sends Default Privileged Information from node data base
No
DECnet-Vax Sends Null Information
to remote node
Inbound Connection (to local node)
from remote node
Explicit Access Control Information Received
Yes
LOGINOUT Uses it
No
Object Data Base Contains Default Inbound Access Control Information
Yes
LOGINOUT Uses it
No
Default Nonprivileged Account Information Associated with Local Node
Yes
LOGINOUT Uses it
No
LOGINOUT Uses No Access Control
ZK-838-82

Figure 2-3: Outbound and Inbound Connect Flowcharts

2.3 USING LOGICAL NAMES

The use of logical names for network operations allows you to refer to network file and task specifications without using actual names that you give these elements. Logical names serve as a kind of shorthand for specifying all or a portion of a full file specification. The inherent flexibility in using logical names allows you to pass file specifications defined at the DCL level to an executing image at run time. For example, logical names allow a program to access local or remote files without changing the program. You can also use logical names to conceal access control information from other users by embedding it in a logical name defined in the process logical name table. Logical names provide convenient and powerful multilevel access control specification.

The rules that govern the use of logical names for network operations are as follows:

- Both the device name and node name elements of a full file specification string can be logical names. However, once a node specification is encountered during file parsing, the device name that follows will be treated as a logical name only if it translates to an equivalence string that was entered in user mode in the process logical name table. Otherwise, the device name is passed unaltered to the remote node, where it is subject to logical name translation.
- A logical name appearing in the device name position can supply any file specification string elements when translated.
- A logical name appearing in the node name position can supply only a node-spec when translated. Therefore, its equivalence string must end with a double colon.
- An access control string associated with a logical node name becomes the new access control string for the node-spec of the equivalence string, even if the node-spec contained an access control string. Thus, you can easily specify a default (or override any) access control string defined for the node-spec resulting from logical name translation.
- After a logical node name is translated, the new node name becomes a candidate for logical node name translation.
- A maximum of ten logical device name translations and ten logical node name translations is permitted. If you exceed these limits, an RMS error (RMS$_LNE) is returned.

Examples of Logical Names

1.
```
$ DEFINE NEW_YORK NYC::

$ DEFINE TORONTO TRNTO::

$ DEFINE FILE TORONTO::DBA1:[INVENTORY.COM]COPYTEST.COM
$ TYPE FILE
```

   This command displays (at the local node) file COPYTEST.COM in directory [INVENTORY.COM] on remote node TRNTO.

2.
```
$ DEFINE A TRNTO::DBA1:[INVENTORY.COM]
$ TYPE A:COPYTEST.COM
```

   This command displays file COPYTEST.COM in an alternate manner.

3.
```
$ DEFINE B TRNTO::
$ TYPE B::DBA1:[INVENTORY.COM]COPYTEST.COM
```

This command displays file COPYTEST.COM in still another manner.

4.
```
$ DEFINE TORONTO TRNTO::
$ DEFINE NODE "TORONTO""TEST RESULTS""::"
$ DEFINE DEVICE NODE::DBA1:
$ DEFINE REMOTE DEVICE:[FINAL.RESULTS]
$ TYPE REMOTE:TEST.DAT
```

This command displays file TEST.DAT in directory [FINAL.RESULTS] on node TRNTO. The file specification was expanded as follows:

```
$ TYPE REMOTE:TEST.DAT

DEVICE:[FINAL.RESULTS]TEST.DAT

NODE::DBA1:[FINAL.RESULTS]TEST.DAT

TORONTO"TEST RESULTS"::DBA1:[FINAL.RESULTS]TEST.DAT

TRNTO"TEST RESULTS"::DBA1:[FINAL.RESULTS]TEST.DAT
```

2.3.1 Iterative Translation

The node name portion of a node specification is translated recursively. For example:

```
$ DEFINE ALPHA BOSTON::
$ DEFINE BETA ALPHA::
$ DEFINE C "BETA""FRED XJ5""::DM1:[TEMP]"
$ TYPE C:FILE.DAT
```

This command displays file FILE.DAT in directory [TEMP] on node BOSTON. The file specification was expanded as follows:

```
$ TYPE C:FILE.DAT

BETA"FRED XJ5"::DM1:[TEMP]FILE.DAT

ALPHA"FRED XJ5"::DM1:[TEMP]FILE.DAT

BOSTON"FRED XJ5"::DM1:[TEMP]FILE.DAT
```

When logical node names are translated iteratively, the access control information first translated overrides subsequent access control information. For example,

```
$ DEFINE TORONTO  "TRNTO""TEST RESULTS""::"
$ DEFINE TEST1  "TORONTO""TEST 1001""::DBA1:"
$ DEFINE TEST2  TORONTO::DBA2:

$ TYPE TEST1:PROC.001,TEST2:PROC.002
```

In the above example, TEST1 translates to TRNTO"TEST 1001"::DBA1: and TEST2 translates to TRNTO"TEST RESULTS"::DBA2:. Note that TORONTO would be an invalid node name were it not a logical name that translated to a node specification containing a node name of one to six characters.

2.3.2 Names Prefixed by an Underscore Character

Device names and node names that are prefixed by an underscore character (_) are not candidates for logical name translation. However, if you prefix the underscore character to a name, it is not considered part of the name (for example, _BOSTON is a valid node name in this respect). For example:

```
$ DEFINE BOSTON TRNTO::
$ TYPE BOSTON::A.DAT,_BOSTON::B.DAT
$ DEFINE/USER DBA0 DBA2:
$ TYPE BOSTON::DBA0:C.DAT,_BOSTON::_DBA0:D.DAT
```

In the example above, A.DAT comes from node TRNTO, B.DAT comes from node BOSTON, C.DAT comes from DBA2 on node TRNTO, and D.DAT comes from DBA0 on node BOSTON. Note that if the logical name DBA0 were not placed in the process table in user mode, it would not have been translated at the local node for the file specification containing C.DAT.

2.4 NETWORK COMMAND TERMINALS

DECnet-VAX network command terminals are implemented via the VAX/VMS remote command terminal facility. This facility permits a single user to establish communication with a remote VAX/VMS node and to use the facilities of that system while physically connected to the local node. By means of this link, you can temporarily become a local user of the remote node and thereby perform functions that the remote node allows its local users to perform.

To establish communication with a remote VAX/VMS node, use the DCL command SET HOST. The format for this command is as follows:

```
SET HOST nodename
```

In this command, nodename is defined as follows:

nodename	is a 1- to 6-character name (or number) specifying the remote node at which you want to log in.

The remote system will prompt for a user name and password, and, if this information is valid, it will cause you to be logged in at the remote node. There is no special control character handling (other than CTRL/Y) for remote command terminal operations. To return control to your local node, type LOGOUT; the following message will appear, indicating that control has been transferred to your local node:

```
%REM-S-END, control returned to node _nodename::
```

NOTE

Repeated pressing of CTRL/Y rapidly will generate a prompt asking if the remote connection should be broken. If you answer "Yes" to the prompt, control will return to the local node. This is useful if for some reason you cannot return to the local node properly.

The following command sequence illustrates the operation of remote command terminals for our network example (the name of the local node is BOSTON):

```
$ SET HOST TRNTO
Username: SMITH
Password:

     Welcome to VAX/VMS Version V3.0 on node _TRNTO::

                    .
                    .
                    .

$ LOGOUT
SMITH  logged out at 8-MAY-82 12:31:55.49

%REM-S-END, control returned to node _BOSTON::

$
                    .
                    .
                    .
```

Once logged in at a remote node, you can use the SET HOST command to establish communication with another node. In the above example, after logging in at node TRNTO, you could type SET HOST DENVER, which would cause you to be logged in at node DENVER. Note that when you are logged out at node DENVER, control returns to node TRNTO. Refer to the VAX/VMS Command Language User's Guide for a complete discussion of the SET HOST command.

CHAPTER 3

REMOTE FILE ACCESS USING DCL

Most VAX/VMS DCL commands allow you to perform file operations at a remote node. These commands enable you to obtain directory listings, manipulate files, and execute command procedures over the network. The DCL commands described in this chapter use VAX-11 RMS to perform the following network file operations:

- List directories located on a remote node
- Copy files to and from remote nodes and between remote nodes
- Append files to a file
- Delete and purge files from a remote node
- Open, read, write, and close files at a remote node from a command procedure
- Submit command procedure files for execution at the remote nodes where they reside
- Print files at the remote nodes where they reside
- Type files located on a remote node
- Sort and merge remote files
- Search remote files
- Obtain file specification or attribute information about remote files
- Compare two files for differences
- Analyze the structure of remote VAX-11 RMS files
- Convert files from one format to another while copying the result to or from a remote node
- Dump the contents of remote files for inspection
- Perform backup operations on remote VAX/VMS disk files

Many VAX/VMS DCL commands permit access to remote files. These commands fall into the following categories: logical name operations, file operations, lexical functions, and record access operations. This chapter defines the commands (and relevant command and file qualifiers) that you can use over the network. The descriptions of the commands include restrictions on the use of certain commands in a heterogeneous network environment (because of features not available on remote systems). For complete descriptions of these commands, consult the VAX/VMS Command Language User's Guide.

3.1 ACCESSING THE NETWORK USING DCL COMMANDS

A VAX/VMS interactive or batch user is able to perform a variety of network file operations through DCL commands. Conceptually, accessing the network at this level is the same as using DCL directly for local operations. For most DCL commands, you need only include a node name as part of the standard VAX/VMS file specification to denote a remote file. In addition, NETMBX and TMPMBX privileges are required to execute most of the commands described in this chapter.

Table 3-1 summarizes the functions of DCL commands that are commonly used to access remote files in a network environment.

Table 3-1: DCL Command Summary

Type of Operation and Command Statement	Function
Logical Name Operations	
ASSIGN	Associates a file specification, node name, or device name with a logical name for subsequent use in commands and programs at the local node
DEASSIGN	Cancels a logical name assignment made with the ASSIGN or DEFINE command
DEFINE	Creates a logical name for use at the local node with an equivalence name string that is a partial or full file specification (similar to the ASSIGN command)
SHOW LOGICAL	Displays the current assignments for logical names and equivalence names made by the ASSIGN or DEFINE command
SHOW TRANSLATION	Searches logical name tables for a specific logical name and displays the equivalence name of the first match found
File Operations	
ANALYZE/RMS_FILE	Analyzes the internal structure of a VAX-11 RMS file, optionally generating an FDL (File Definition Language) file

(continued on next page)

Table 3-1 (Cont.): DCL Command Summary

Type of Operation and Command Statement	Function
APPEND	Adds the contents of one or more files to the end of another file
BACKUP	Performs save and restore operations on local files using a saveset residing on a remote VAX/VMS node.
CONVERT	Copies records from one file to another file, changing the organization and record format to that of the second file (if it exists) or creating a new file using the file attributes specified in an FDL file
COPY	Copies one or more files to or from a remote node into one or more additional files
CREATE	Creates a sequential disk file from records that follow the command in the input stream
DELETE	Deletes one or more remote files
DIFFERENCES	Compares the contents of two files and produces an output file that lists any differences found
DIRECTORY	Displays the file name and optional file attribute information about a remote file or group of remote files
DUMP/RECORDS	Displays the contents of a remote file in the data format specified
MERGE	Combines two or more similarly sorted remote files into one new file
PRINT/REMOTE	Queues for printing one or more files at the nodes where they reside
PURGE	Purges one or more remote files
SEARCH	Searches one or more files and lists all occurrences of one or more specified strings

(continued on next page)

Table 3-1 (Cont.): DCL Command Summary

Type of Operation and Command Statement	Function
SORT	Reorders records in a remote file and creates a new output file (or an address file to access the records)
SUBMIT/REMOTE	Queues for execution one or more command procedures at the nodes where they reside
TYPE	Displays the contents of a remote file or files
Lexical Functions	
F$FILE_ATTRIBUTES	Returns attribute information about a remote file
F$PARSE	Returns a partial or full file specification for a remote file
F$SEARCH	Returns the full file specification for the next remote file that matches the given wild card file specification
Record Access Operations	
CLOSE	Closes a remote file previously opened by the OPEN command
OPEN	Opens a remote file for reading or writing at the command level
READ	Reads a single record from a remote input file
WRITE	Writes a single record to a remote output file

3.2 LOGICAL NAME COMMANDS

Several DCL commands permit you to create, delete, and display logical names. Although logical name manipulation is performed locally by the commands described in this chapter, use of logical names in file specifications in other DCL commands does affect the network. Consequently, the logical name support commands are described below.

3.2.1 ASSIGN, DEASSIGN, and DEFINE

The ASSIGN, DEASSIGN, and DEFINE commands allow you to generate logical names for nodes and devices for use in file specifications.

These commands provide a convenient way to use logical file specifications without having to define physical device specifications. When used for network operations, these commands support all command and file qualifiers that you would normally use locally.

Examples

1. `$ DEFINE TORONTO TRNTO::DBA0:[DECNET.DEMO.COM]`

   This DEFINE command places the logical name TORONTO in the process logical name table with an equivalence name of TRNTO::DBA0:[DECNET.DEMO.COM].

2. `$ DEFINE LOCAL "BOSTON""JOHN_SMITH JKS""::"`

   This DEFINE command places the logical name LOCAL in the process logical name table with a remote node equivalence name of BOSTON"JOHN_SMITH JKS"::. To satisfy conventions for local DCL command string processing, you must use three sets of quotation marks, so that access control information will be enclosed in one set of quotation marks in the equivalence name.

3. `$ ASSIGN DALLAS::DB0:  DATA`

   This ASSIGN command associates the logical name DATA with the device specification DB0 on remote node DALLAS. Subsequent references to the logical name DATA result in references to the disk on the remote node.

4. `$ DEASSIGN DATA`

   This DEASSIGN command cancels the logical name assignment made in the above example.

3.2.2 SHOW LOGICAL and SHOW TRANSLATION

The SHOW LOGICAL command displays current logical name assignments and the SHOW TRANSLATION command displays the result of translating a logical name. For a discussion of the use of logical names for network operations, see Chapter 2.

Examples

1. `$ SHOW LOGICAL`

   This SHOW LOGICAL command displays the current contents of the process, group, and system logical name tables.

2. `$ SHOW TRANSLATION MASTER`

   This command causes the logical name tables to be searched for the logical name MASTER, and displays its current equivalence name.

3.3 COMMANDS FOR FILE HANDLING

Many DCL commands that contain file specifications can be used to access files stored on remote nodes. The following is a list of DCL commands that are useful in a network context and are supported in part or in full in that environment. This list notes restrictions on using certain command qualifiers and file qualifiers when entering particular commands in a network context. Complete descriptions of the file-handling commands appear in the VAX/VMS Command Language User's Guide.

```
ANALYZE/RMS_FILE file-spec[,...]
```

This command is supported only for the examination of files generated by VAX-11 RMS or RMS-11.

```
APPEND input-file-spec[,...] output-file-spec
```

```
BACKUP input-specifier output-specifier
```

This command is supported only to access savesets located on remote VAX/VMS nodes. An input or output specifier that includes a remote node name must also include the file qualifier /SAVE_SET.

The copy, compare, and journal operations are not supported.

```
CONVERT input-file-spec[,...] output-file-spec
```

```
COPY input-file-spec[,...] output-file-spec
```

The following file qualifiers are not supported if the output file is on a remote node:

```
/[NO]OVERLAY
/[NO]REPLACE
```

```
CREATE file-spec
```

The /DIRECTORY qualifier is not supported.

```
DELETE file-spec[,...]
```

```
DIFFERENCES master-file-spec [revision-file-spec]
```

```
DIRECTORY [file-spec[,...]]
```

The command qualifier /FULL is supported except for the file identification number which is displayed as <unknown>.

```
MERGE input-file-spec1,input-file-spec2[,...] output-file-spec
```

```
DUMP/RECORDS[=(option[,...])] file-spec
```

The following command qualifiers are not supported:

```
/ALLOCATED
/BLOCKS
```

```
PRINT/REMOTE file-spec[,...]
```

No other qualifiers may be used with /REMOTE.

```
PURGE file-spec[,...]


SEARCH file-spec[,...] search-string[,...]


SORT input-file-spec[,...] output-file-spec

   The /RSX11 qualifier is not supported.


SUBMIT/REMOTE file-spec[,...]

   No other qualifiers may be used with /REMOTE.


TYPE file-spec[,...]
```

The DCL commands listed below are not supported for access to files on remote nodes:

```
@
RENAME
RUN
SET DEFAULT
UNLOCK
```

The following subsections describe in more detail the use of DCL commands for handling files over the network. Note that if you do not specify an access control string in the file specification in a DCL command, the default DECnet account at that node is accessed if it exists.

3.3.1 ANALYZE/RMS_FILE

Use the ANALYZE/RMS_FILE command to analyze the internal structure of a remote VAX-11 RMS or RMS-11 file. You can specify the command qualifier /FDL to generate an FDL (File Definition Language) file from the data file. Using other command qualifiers, you can check the file structure for errors, generate a statistical report on the file's structure and use, or enter interactive mode to explore the structure of the file. You can specify only one of these command qualifiers in each command.

Examples

1.
```
$ ANALYZE/RMS_FILE DENVER::DB1:[PROD]RUN.DAT
```

   This ANALYZE/RMS_FILE command analyzes the structure of the file RUN.DAT residing at remote node DENVER.

2.
```
$ ANALYZE/RMS_FILE/FDL/OUTPUT=TEST.FDL
$_File(s):  DENVER::DB1:[PROD]RUN.DAT
```

   This ANALYZE/RMS_FILE command analyzes the structure of the file RUN.DAT at remote node DENVER and generates the FDL file TEST.FDL at the local node.

3.3.2 APPEND and COPY

Use the APPEND command to add the contents of one or more specified input files to the end of a specified output file. Use the COPY command to create a new file from one or more existing files.

Examples

1.
```
$ COPY BOSTON::DMA2:TEST.DAT;5
$_To:   TRNTO::DBA1:[MODEL.TEST]TEST.DAT/ALLOCATION=50
```

   This COPY command copies the file TEST.DAT;5 on device DMA2 at node BOSTON to a new file named TEST.DAT at remote node TRNTO. The /ALLOCATE qualifier initially allocates 50 blocks for the new file TEST.DAT at node TRNTO.

2.
```
$ APPEND/LOG BOSTON"JOHN_SMITH JKS"::DEMO1.DAT,DEMO2.DAT
$_To:   TRNTO::DBA1:[MODEL.TEST]TEST.DAT
```

   This APPEND command adds the contents of the files DEMO1.DAT and DEMO2.DAT at remote node BOSTON to the end of file TEST.DAT at remote node TRNTO. The /LOG qualifier displays the fully expanded names of the files used.

3.
```
$ COPY SAMPLE.EXE DALLAS::DB0:[117,10]SAMPLE.EXE/CONTIGUOUS
```

   This COPY command copies the file SAMPLE.EXE on the local node to a file with the same name at remote node DALLAS. The /CONTIGUOUS qualifier indicates that the output file is to occupy consecutive physical disk blocks.

4.
```
$ COPY DALLAS::T1.DAT,T2.DAT,T3.DAT *.*
$ COPY *.* TRNTO::*.*
```

   The first COPY command copies the three files T1.DAT, T2.DAT, and T3.DAT on remote node DALLAS to the local node while preserving the names of the files. The second example is a more generalized form of the COPY command. All files within the user directory at the local node are copied to the remote node TRNTO. The new files will have the same names as the input files.

3.3.3 BACKUP

You can use the BACKUP command to save local files in a BACKUP saveset residing on a remote VAX/VMS node. You can also use this command to restore at the local node files that were previously saved in a saveset on a remote VAX/VMS node. Use BACKUP/LIST to display the names and attributes of files cataloged in a remote saveset. The remote BACKUP saveset cannot be on magnetic tape; it must reside on disk.

Examples

1.
```
$ BACKUP
$_From: DB1:[SCHED]*.*
$_To:    DENVER::DBA2:[SAVE]SCH.BCK/SAVE_SET
```

   This BACKUP command saves the files in the directory SCHED on disk DB1 at the local node in the BACKUP saveset SCH.BCK at remote node DENVER. The /SAVE_SET qualifier is required to identify the output specifier as a saveset on a Files-11 medium.

2. `$ BACKUP/LIST DENVER::DBA2:[SAVE]SCH.BCK/SAVE_SET`

   This BACKUP command lists the BACKUP summary information, the original BACKUP command used, and the file name, size and creation date for each file in the saveset created in example 1. The /SAVE_SET qualifier is required to identify the input specifier as a saveset on a Files-11 medium.

3.3.4 CONVERT

Use the CONVERT command to transfer records from a source data file to a second data file, which can differ in file organization and format from the first. You can use this command to transfer files to or from a remote node while altering file attributes. If the output file exists, the Convert Utility (CONVERT) changes the organization and format of the data from the input file to that of the output file. If the output file does not exist, the Convert Utility creates it from the file attributes specified in an FDL (File Definition Language) file. You can also use the CONVERT command to copy files to a remote node or to retrieve them without modifying file attributes. However, CONVERT transfers the file record by record and thus does not use block I/O.

The Convert Utility is described in the VAX-11 Record Management Services Utilities Reference Manual.

Examples

1. `$ CONVERT/FDL=TEST.FDL TRNTO::DBA1:[EXP]SUB.DAT CUM.DAT`

   This CONVERT command creates a new sequential file CUM.DAT with stream record format at the local node, according to the specification in the previously created FDL file, TEST.FDL. The input file SUB.DAT at remote node TRNTO is sequential with variable-length record format. The Convert Utility copies records from SUB.DAT to CUM.DAT, changing the format of the records.

   The contents of the FDL file TEST.FDL are as follows:

```
SYSTEM
        SOURCE              vax/vms

FILE
        ORGANIZATION        sequential

RECORD
        BLOCK_SPAN          yes
        CARRIAGE_CONTROL    carriage_return
        FORMAT              stream
        SIZE                0
```

2. `$ CONVERT MASTER.DAT DENVER::DB1:[PROD]MASTER.SAV`

   This CONVERT command creates a new file called MASTER.SAV at remote node DENVER from the file MASTER.DAT at the local node. Because the /FDL qualifier is not used, the new file has the same file organization and record format as the original file. The action of this CONVERT command is similar to the function performed by the COPY command. However, CONVERT transfers the file record by record and thus does not use block I/O.

3. `$ CONVERT/APPEND SALES.TMP KANSAS::[200,2]SALES.CMD`

   This CONVERT command causes records from the file SALES.TMP at the local node to be added sequentially to the end of the output file SALES.CMD at remote node KANSAS. The file SALES.TMP is sequential with variable-length record format, and the file SALES.CMD is sequential with stream record format. When the Convert Utility loads records from the input file to the output file, it changes the record format.

3.3.5 CREATE

Use the CREATE command to create sequential disk files on a remote node.

Example

```
$ CREATE TRNTO::DBA1:[MODEL.TEST]TEST.DAT
1
22
333
4444
^Z
$
```

The CREATE command creates a sequential file named TEST.DAT that consists of the characters entered on the lines following the CREATE command. The CTRL/Z entry indicates the end of the file.

3.3.6 DELETE and PURGE

Use the DELETE command to delete one or more files from a mass storage volume on a remote node. The DELETE command requires that an explicit version number be included in a file specification unless the file specification is delimited by quotation marks. A null version number (;) or a version number of zero (;0) implies the highest version of the file. Use the PURGE command to delete all but the highest-numbered version or versions of one or more files residing at remote nodes.

Examples

1.
   ```
   $ DELETE/LOG
   $_File:  TORONTO::DBA0:[100,5]WORKORDER.DAT;3,OUTPUT.FIL;2
   ```

   This DELETE command deletes the files WORKORDER.DAT;3 and OUTPUT.FIL;2 from device DBA0 at remote node TORONTO. The /LOG qualifier displays the file specification of each file deleted. Note that TORONTO is a logical name.

2. `$ DELETE DALLAS"FRED R2D2"::DK0:[305,321]DECODE.LIS;1`

   This DELETE command deletes the file DECODE.LIS;1 in directory [305,321] on device DK0 at remote node DALLAS.

3.
   ```
   $ DELETE/CONFIRM
   $_File:  TRNTO::[SAM.OBJ]A.OBJ;,A.EXE;,[SAM.LIS]A.LIS;
   ```

   This DELETE command queries the user whether or not each of the successive files on remote node TRNTO should be deleted.

4.
   ```
   $ DELETE QUEBEC::"DX1:DEAL.BIG"
   $ DELETE QUEBEC::DX1:DEAL.BIG;
   ```

   Both of these DELETE commands delete the file DEAL.BIG on device DX1 at remote node QUEBEC. Note that the DELETE command requires an explicit version number in a file specification. The file to be deleted is on a remote node whose file syntax does not recognize version numbers. (QUEBEC is an RT-11 node.) Therefore, the file specification should be enclosed in quotation marks or entered with a null version number (that is, a trailing semicolon).

5.
   ```
   $ PURGE TRNTO::DBA1:[EXAMPLE]*.LIS/KEEP=2
   ```

   This PURGE command deletes all but the two highest-numbered versions of each file of the type LIS in the directory EXAMPLE on remote node TRNTO.

3.3.7 DIFFERENCES

Use the DIFFERENCE command to compare the contents of two files (either of which can be local or remote) on a record-by-record basis. The command produces an output file listing any differences.

Examples

1.
   ```
   $ DIFFERENCES BOSTON::DBA2:TEST.DAT TRNTO::DBA1:[PGM]TEST.DAT
   ```

   This command compares two remote files and displays any differences found. The first file is TEST.DAT on remote node BOSTON and the second file is TEST.DAT on remote node TRNTO.

2.
   ```
   $ DIFFERENCES BOSTON::TEST.DAT
   ```

   This command compares the two highest versions of the file TEST.DAT in the nonprivileged default DECnet account on remote node BOSTON.

3.3.8 DIRECTORY

Use the DIRECTORY command to list files and their attributes in a directory on a remote node.

Examples

1.
   ```
   $ DIRECTORY TRNTO::DBA1:[DOE]LOGIN.COM
   ```

   This DIRECTORY command lists all versions of the file LOGIN.COM under directory DOE at remote node TRNTO.

2.
   ```
   $ DIRECTORY/DATE/SIZE=ALL TRNTO::DBA1:[DOE...]*.COM
   ```

   This DIRECTORY command lists all versions of all files with a file type of COM in all subdirectories of [DOE] on remote node TRNTO. The listing includes the creation date with each file, and the file size both in blocks used and in blocks allocated for each file.

3.
```
$ DIRECTORY/FULL BOSTON::*VAX*.*
```

   This DIRECTORY command displays full directory information for each file whose file name contains the string "VAX" in the nonprivileged default DECnet account on node BOSTON.

4.
```
$ DIRECTORY/SINCE=TODAY BOSTON"JOHN_SMITH JKS"::[.MEMO]W%%
```

   This DIRECTORY command lists each file created today in the user's subdirectory MEMO whose file name begins with "W" and contains three characters (for example, W03.DOC, WWW.TMP).

5.
```
$ DIRECTORY TORONTO::
```

   This DIRECTORY command lists all the files cataloged in the directory associated with the default account being accessed at remote node TORONTO. Note that TORONTO is a logical name.

3.3.9 DUMP/RECORDS

Use the DUMP/RECORDS command to display the contents of remote files in ASCII, hexadecimal, decimal, or octal representation. The DUMP command qualifiers /ALLOCATED and /BLOCKS are not supported in the network context.

Example

```
$ DUMP/RECORDS/OCTAL/WORD
$_File:  DALLAS::DB0:[117,10]CALC.DAT/PRINTER
```

This DUMP/RECORDS command dumps the contents of the file CALC.DAT, which resides at remote node DALLAS; formats the output both in octal words and in character strings; and queues the output to the system printer at the local node.

3.3.10 PRINT/REMOTE

Use the PRINT/REMOTE command to queue files for printing at the remote nodes on which they exist. One copy of each file specified is printed. You can specify on the same command line files that exist at different nodes. If you specify in a PRINT/REMOTE command two or more files that reside on the same remote VAX/VMS node, each file is entered in the SYS$PRINT queue as a separate print job. (Note that the PRINT/REMOTE command does not copy the files to the remote node; a separate COPY command must be issued if the file does not reside at the remote node on which it is to be printed.)

The /REMOTE qualifier is required in the PRINT command whenever a file specification contains a node name. When you specify /REMOTE, you cannot specify any other qualifiers for the command. The /REMOTE qualifier can appear with the command or after a file specification. The PRINT command supplies a default file type of LIS if you omit the file type from the file specification.

Examples

1.
```
$ PRINT/REMOTE BOSTON::WORK$:[DECNET.V3]USRGUIDE.MEM,EXP1.FOR
$ PRINT BOSTON::WORK$:[DECNET.V3]USRGUIDE.MEM,EXP1.FOR/REMOTE
```

   Either of the two commands shown above can be entered at node TRNTO to queue for printing at node BOSTON the files USRGUIDE.MEM and EXP1.FOR which reside at node BOSTON. The files are entered in the SYS$PRINT queue as separate print jobs.

2.
```
$ COPY REPORT.MEM BOSTON::*.*
$ PRINT/REMOTE BOSTON::REPORT.MEM
```

   The two commands shown above are entered at node TRNTO to cause the file REPORT.MEM located at node TRNTO to be printed at remote node BOSTON. The file is copied into the default DECnet directory at the remote node and is not deleted after printing.

3.
```
$ COPY REPORT.MEM BOSTON::LPA0:
```

   An alternative way of performing the same operation as in example 2 above is to copy the file REPORT.MEM at node TRNTO to the printing device on the remote system. If the printing device is spooled (as is usually the case for line printers on a VAX/VMS system), then the file will not be sent directly to the device, but rather will be temporarily stored on disk, entered into the print queue for the device, and deleted after it is printed.

3.3.11 SEARCH

Use the SEARCH command to search one or more remote files for a specified string or strings.

Example

```
$ SEARCH TRNTO::DBA1:[EXP]SUB.DAT,DATA.LIS
$_String(s):  NAME
```

The SEARCH command causes the files SUB.DAT and DATA.LIS at remote node TRNTO to be searched for all occurrences of the character string NAME. The list of all occurrences of NAME is printed at the local terminal.

3.3.12 SORT and MERGE

Use the SORT command to invoke the VAX-11 Sort Utility. This program reorders records in the input file as directed and creates a new output file or, optionally, an address file that you can use to access the reordered records.

Use the MERGE command to invoke the VAX-11 Merge Utility, which combines two or more sorted files into a single output file that the utility program creates. The files to be combined must be similarly sorted, but can reside at different VMS nodes.

Examples

1.
```
$ SORT/KEY=(POSITION:1,SIZE:7) -
$_DENVER::DB1:[RECS]RNDM.FIL ALPHANM.SRT/KEY=(1,7)
```

   This SORT command requests a default alphanumeric sort of the records in the file RNDM.FIL at remote node DENVER. The SORT program sorts the records on the basis of the contents of the first seven characters in each record and writes the sorted list into the output file ALPHANM.SRT created in the default directory at the local node.

2.
```
$ MERGE/KEY=(POSITION:1,SIZE:30) -
$_TRNTO:[PGM]FILE1.SRT,FILE2.SRT/CHECK_SEQUENCE -
$_MERGEFILE.DAT
```

   This MERGE command causes two identically sorted files, FILE1.SRT and FILE2.SRT, on the directory PGM at remote node TRNTO to be merged into another file, MERGEFILE.DAT, created in the current default directory at the local node. The input file qualifier /CHECK_SEQUENCE is specified to ensure that the input files are sorted in the correct order.

3.3.13 SUBMIT/REMOTE

Use the SUBMIT/REMOTE command to enter command procedure files residing on a remote node into the batch job queue for execution at the remote node. You can specify on the same command line command procedure files located at different remote nodes. If you specify in a SUBMIT/REMOTE command two or more files located at the same remote VAX/VMS node, each file is entered in the SYS$BATCH queue as a separate batch job. (Note that the SUBMIT/REMOTE command does not copy the files to the remote node: a separate COPY command must be issued if the file does not reside at the remote node where it is to be executed.)

The /REMOTE qualifier is required in the SUBMIT command whenever a file specification contains a node name. When you specify /REMOTE, you cannot specify any other qualifiers for the command. The /REMOTE qualifier can appear with the command or after a file specification. If you omit the file type from the file specification, the SUBMIT command supplies a default file type of COM.

Examples

1.
```
$ SUBMIT/REMOTE BOSTON::DMA3:[BROWN]JOBS.COM,LISTALL.COM
$ SUBMIT BOSTON::DMA3:[BROWN]JOBS.COM,LISTALL.COM/REMOTE
```

   This SUBMIT/REMOTE command entered at node TRNTO submits the files JOBS.COM and LISTALL.COM on device DMA3 at remote node BOSTON for execution as separate batch jobs.

2.
```
$ COPY ANALYSIS.COM BOSTON::*.*
$ SUBMIT/REMOTE BOSTON::ANALYSIS.COM
```

   The two commands shown above are entered at node TRNTO to cause the file ANALYSIS.COM residing at node TRNTO to be executed at remote node BOSTON. The file is copied into the default DECnet directory at the remote node, and is not deleted after execution.

3.3.14 TYPE

Use the TYPE command to display the contents of one or more remote files on the current output device. If you omit the file type in the file specification, the TYPE command supplies a default of LIS.

Examples

1. `$ TYPE TRNTO::DBA1:[DOE]LOGIN.COM`

   The TYPE command requests that the file LOGIN.COM in directory DOE at remote node TRNTO be displayed at the local terminal.

2. `$ TYPE KANSAS::"$TEXT.CMD"`

   The TYPE command requests that the file TEXT.CMD on remote RSTS/E node KANSAS be displayed at the local terminal. Note that you use quotation marks when the file specification syntax of the remote node differs from that of VAX/VMS.

3. `$ TYPE TORONTO::NOTICE.TXT/OUTPUT=TEMP.TXT`

   The TYPE command requests that the file NOTICE.TXT at the remote node designated by the logical name TORONTO be written to the specified output file, TEMP.TXT, on the local node rather than to SYS$OUTPUT.

3.4 LEXICAL FUNCTIONS

DCL command procedure files can include lexical functions that return information about remote files. The lexical functions that can be used in a network environment are:

```
F$FILE_ATTRIBUTES(file-spec,item)

F$PARSE(file-spec[,default-spec][,related-spec][,field])

F$SEARCH(file-spec[,stream-id])
```

Descriptions of the use of these lexical functions in returning information on remote files is given below. Complete descriptions of these lexical functions appear in the _VAX/VMS Guide to Using Command Procedures_.

An example of a command procedure that includes all three lexical functions appears in Section 3.6.1.

3.4.1 F$FILE_ATTRIBUTES

Use the F$FILE_ATTRIBUTES lexical function in a command procedure to return a particular item of attribute information about a specified local or remote file. Listed under the description of F$FILE_ATTRIBUTES in the _VAX/VMS Guide to Using Command Procedures_ are the items you can specify in the function (for example, ORG for file organization). The file-spec is specified as a string expression, or a symbol equated to a string expression; no wild card characters are allowed.

Example

```
$ RFM = F$FILE("KANSAS::SY:[200,2]SALES.CMD","RFM")
$ SHOW SYMBOL RFM
  RFM = "STM"
```

This example returns the record format string of STM (stream) for the file SALES.CMD at remote RSTS/E node KANSAS.

3.4.2 F$PARSE

Use the F$PARSE lexical function to return a full file specification, or a particular field of that specification, for a local or remote file. To identify the file in the lexical function, specify a file-spec and, optionally, a default-spec and related-spec. These arguments are string expressions or symbols equated to string expressions.

To obtain a portion of a file specification, specify a field name in the lexical function. The field names can be any of the following: DEVICE, DIRECTORY, NAME, NODE, QUOTED, TYPE, or VERSION.

Example

```
$ SPEC = F$PARSE("DENVER::DB1:[PROD]RUN.DAT",,,"TYPE")
$ SHOW SYMBOL SPEC
  SPEC = ".DAT"
```

In this example, the F$PARSE lexical function returns the file type DAT for the file RUN.DAT at remote node DENVER.

3.4.3 F$SEARCH

Use the F$SEARCH lexical function to obtain full file specifications for local or remote files that match the file-spec given in the lexical function. The file-spec, a string expression or symbol equated to a string expression, can be any valid VAX/VMS file specification, and can include null or wild card fields. Each consecutive search function returns the next matching file specification in sequence. When there are no more resultant strings, the null string is returned.

Example

```
$LOOP:
$ FILESPEC = F$SEARCH("TRNTO::DBA1:[PROD]*.DAT")
$ IF FILESPEC .EQS. "" THEN EXIT
  .
  .
$ GOTO LOOP
```

This example causes the directory [PROD] at remote node TRNTO to be searched for all files of the type DAT.

3.5 COMMANDS FOR ACCESSING RECORDS

You can use DCL commands within command procedures to open and close files that reside on remote nodes and to read and write records in these files. In command procedure files to be executed during network operations, you can specify all command and file qualifiers for OPEN, CLOSE, READ, and WRITE that you would normally use in command procedures to access local files.

3.5.1 OPEN and CLOSE

Use the OPEN command to open a file for reading or writing at the command level. The CLOSE command closes a file that was opened for input or output with the OPEN command and deassigns the logical name specified when the file was opened.

The DCL statement OPEN/WRITE creates a file in VFC (variable with fixed control) format. If the remote node does not support VFC format, an RMS$_SUPPORT error will be returned.

Example

```
$ OPEN/READ  INPUT_FILE  TRNTO::DBA0:[COST]INVENTORY.DAT
$READ_LOOP:
$ READ/END_OF_FILE=ENDIT         INPUT_FILE       NUM
$ FIRST_CHAR=F$EXTRACT(0,1,NUM)
$ WRITE SYS$OUTPUT FIRST_CHAR
$ GOTO  READ_LOOP
$ENDIT:
$ CLOSE INPUT_FILE
```

This command procedure opens the file INVENTORY.DAT located at remote node TRNTO as an input file and assigns it the logical name INPUT_FILE. The READ command reads a record from the logical file INPUT_FILE into the symbol named NUM. The next two commands extract the first character from the record and write the record to the SYS$OUTPUT device. These two steps occur for all records of the file until the procedure reaches the end-of-file. At this point, the CLOSE command closes the file and deassigns the logical name INPUT_FILE.

3.5.2 READ and WRITE

Use the READ command to read a single record from a specified remote input file. Use the WRITE command to write a record to a specified output file.

Example

1.
```
$ OPEN/WRITE   OUTPUT_FILE   TRNTO::DBA1:[PGM]PLAN.DAT
$ WRITE OUTPUT_FILE "BEGINNING PHASE 3"
```

   This WRITE command writes a single line of text to the file PLAN.DAT at remote node TRNTO.

2.
```
$ OPEN/READ      INPUT_FILE       TRNTO::INVENTORY.DAT
$ OPEN/WRITE     OUTPUT_FILE      RECEIVE.DAT

$ READ   INPUT_FILE   DATA_LINE
$ WRITE  OUTPUT_FILE  DATA_LINE
```

   The READ command requests data from the file INVENTORY.DAT at remote node TRNTO. The WRITE command writes the value of the symbol DATA_LINE to the local file RECEIVE.DAT.

3.6 COMMAND PROCEDURE EXAMPLES

The following examples illustrate DCL command procedure files you can use in a network environment.

3.6.1 Command Procedure Using Lexical Functions

The command procedure shown below, called LISTIDX.COM, employs the lexical functions F$PARSE, F$SEARCH, and F$FILE_ATTRIBUTES to locate indexed files in a directory at a local or remote node.

```
        LISTIDX.COM
$ !
$ !     This command procedure displays the names of all indexed
$ !     files found in the specified directory, excluding files
$ !     with a file type of TMP.  P1 is a file-spec that can
$ !     optionally be used to indicate the directory to be
$ !     searched.
$ !
$ FILE = F$PARSE(P1,"*.*")
$ WRITE SYS$OUTPUT "A list of indexed files follows ..."
$ WRITE SYS$OUTPUT ""
$LOOP:
$ NEXT = F$SEARCH(FILE)
$ IF NEXT .EQS. "" THEN GOTO DONE
$ IF F$PARSE(NEXT,,,"TYPE") .EQS. ".TMP" THEN GOTO LOOP
$ IF F$FILE_ATTRIBUTES(NEXT,"ORG") .NES. "IDX" THEN GOTO LOOP
$ WRITE SYS$OUTPUT NEXT
$ GOTO LOOP
$DONE:
$ EXIT
```

3.6.2 Command Procedure Using SYS$NET

The example below illustrates a command procedure, called SHOWBQ.COM, that returns batch job status information to its requestor. Note that you can use SHOWBQ.COM for task-to-task communication by entering a task-spec-string in a TYPE command. For example:

```
$ TYPE TRNTO"BROWN JUNE"::"TASK=SHOWBQ"
```

In this command procedure, SYS$OUTPUT is equated to SYS$NET in user mode to allow the SHOW QUEUE image to communicate over the logical link by opening SYS$OUTPUT. When the SHOW QUEUE image exits, the temporary definition of SYS$OUTPUT is deleted. In other words, only one DCL image can use the logical link as the communication path to the requestor at the other node.

```
        SHOWBQ.COM
$ !
$ !     This  command procedure  returns status information about
$ !     Jobs  entered in  batch  queues  on  the  system where it
$ !     executes.   It  may  be   run  interactively as  a command
$ !     procedure,  submitted  as a  local or remote batch job, or
$ !     invoked  as a  'remote task'  to display information about
$ !     another system.  For example:
$ !
$ !     $ @SHOWBQ
$ !     $ SUBMIT SHOWBQ
$ !     $ SUBMIT/REMOTE node::SHOWBQ
$ !     $ TYPE node::"TASK=SHOWBQ"
$ !
$ IF F$MODE() .EQS. 'NETWORK' THEN GOTO NET
$ SHOW QUEUE/BATCH/BRIEF/ALL
$ EXIT
$NET:
$ DEFINE/USER SYS$OUTPUT SYS$NET
$ SHOW QUEUE/BATCH/BRIEF/ALL
$ PURGE/KEEP=1 SYS$LOGIN:SHOWBQ.LOG
$ EXIT
```

3.7 DISPLAY OF ERROR MESSAGES IN NETWORK ENVIRONMENT

When you enter a DCL command to perform a network file operation that does not complete successfully, one or more error messages are written to SYS$ERROR. The following sequence of error messages is typical:

1. An error message generated by the DCL command interpreter

2. A primary error message generated by the VAX-11 Record Management Facility (RMS)

3. An optional secondary error message associated with the primary RMS error (from a facility involved in the network file operation)

Network-specific RMS completion codes and their corresponding message text are described in Section 4.3.

Examples

1.
```
$ COPY BOSTON::DBB2:[TEST]RSLT.DAT *.*
%COPY-E-CLOSEIN, error closing _BOSTON::DBB2:[TEST]RSLT.DAT;1
as input
-RMS-E-CRC, network DAP level CRC check failed
```

   A file-level CRC checksum error was detected when the input file RSLT.DAT was closed. Error messages generated by the DCL command interpreter and the RMS facility are displayed on the terminal.

2.
```
$ COPY INDEX.DAT BANGOR::TEMP.DAT
%COPY-E-OPENOUT, error opening _BANGOR::TEMP.DAT;  as output
-RMS-F-SUPPORT, network operation not supported
-FAL-F-ORG, file organization field rejected
```

   An attempt to copy the file INDEX.DAT to TEMP.DAT at remote node BANGOR failed because the latter does not support indexed files. The following generated error messages: the DCL command interpreter, the RMS facility, and the remote File Access Listener (FAL) file server utility.

CHAPTER 4

REMOTE FILE ACCESS USING RMS

VAX/VMS provides an efficient and flexible means for accessing remote files in a network environment. Using VAX-11 RMS facilities, you can perform file-handling operations on entire files or individual records via programmed calls in VAX-11 MACRO or in one of the higher-level languages supported over the network. The programming procedures described in this chapter use standard VAX-11 RMS and higher-level language I/O calls to:

- Create and delete remote files
- Process existing remote files
- Read, write, update, or delete individual records within remote files
- Perform miscellaneous operations on a file such as rewinding a record stream, displaying or modifying file attributes, or extending the size of a file

This chapter describes the general procedures for accessing remote files using VAX-11 MACRO. Specifically, it discusses network access at the RMS level, network restrictions on file access, network error reporting, and the use of higher-level and MACRO languages in accessing remote files. The information and examples presented herein also provide the necessary framework for the discussion of remote file access found in each higher-level language user guide. You should also be familiar with the VAX-11 Record Management Services Reference Manual.

4.1 ACCESSING THE NETWORK AT THE RMS LEVEL

Conceptually, accessing the network at the RMS level is the same as accessing RMS directly for local file-handling operations. To access remote files on a VAX/VMS node, use either DCL commands or VAX-11 RMS service calls along with a standard VAX/VMS file specification that includes a node name.

Because higher-level language I/O calls are translated into VAX-11 RMS calls, the term "RMS service calls" in this chapter includes these language-specific calls.

For remote file processing, VAX-11 RMS integrates the network software necessary to translate standard RMS service calls into the appropriate system service calls, thereby providing a transparent user interface to the network.

When you issue an RMS service call with a file specification that specifies a remote node, VAX-11 RMS communicates the access request via the Data Access Protocol (DAP) to the File Access Listener (FAL) task at the remote node. Each DECnet node that supports remote file access has a FAL task that receives and processes remote file access requests. FAL translates the calls of the accessing process into system-specific file-handling calls to perform the desired operations on files at that node. Figure 4-1 illustrates this entire process as it relates to file access processing in both DCL and VAX-11 RMS.

The way you access individual files on a remote system depends on the source language of the accessing program and on the file system that resides on the remote node. The next section of this chapter discusses these considerations as they pertain to programming remote file-handling applications.

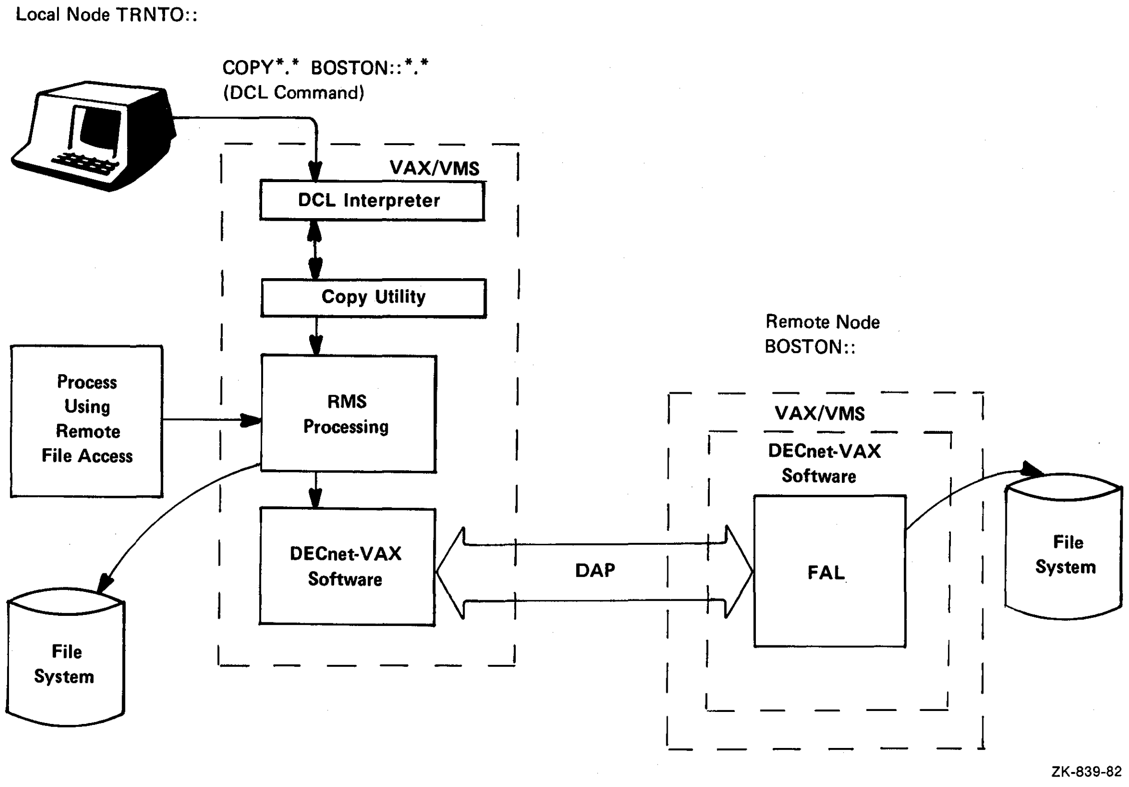


Figure 4-1: Remote File Access (DCL and RMS)

4.2 VAX-11 NETWORK FILE ACCESS RESTRICTIONS

VAX/VMS supports transparent remote file and record access at the programming level, including the use of most VAX-11 RMS file and record operations normally available to the VAX/VMS programmer. Within the context of these operations, however, certain restrictions apply to the method of file and record access you can use. Specifically, each programming language specifies the way you can access individual files. In addition, the way you access a file may be restricted further by the type of file operations supported by the remote file system. The user guide for each higher-level language describes those restrictions that apply for file access operations performed in that language. VAX-11 RMS restrictions are listed below and in Section 4.5.3.

Table 4-1 summarizes the file organizations and record formats that VAX-11 RMS defines for network operations on remote files from a local VAX/VMS node. Table 4-2 shows the RMS record access methods and block I/O modes for such network operations.

Table 4-1: VAX-11 RMS File and Record Characteristics for Network Operations

File Organization	Record Format					
	FIX[1]	VAR[2]	VFC[3]	STM[4]	STMCR[5]	STMLF[6]
Sequential	Yes	Yes	Yes	Yes	No	No
Relative	Yes	Yes	Yes	NA[7]	NA	NA
Indexed	Yes	Yes	NA	NA	NA	NA

1 - FIX:	Fixed-length record format
2 - VAR:	Variable-length record format
3 - VFC:	Variable-length with fixed-length control record format
4 - STM:	Stream format with record terminator set of LF, FF, VT, and CRLF
5 - STMCR:	Stream format with record terminator set of CR
6 - STMLF:	Stream format with record terminator set of LF
7 - NA:	Not applicable

Table 4-2: VAX-11 RMS Access Modes for Network Operations

File Organization	Record Access Mode				Block I/O Mode	
	Sequential	Random by:			Sequential	Random by Virtual Block Number
		Relative Record Number	Key Value	Record File Address		
Sequential	Yes	Yes[1]	NA	Yes	Yes	Yes
Relative	Yes	Yes	NA	Yes	Yes	Yes
Indexed	Yes	NA	Yes	Yes	Yes	Yes

1. For fixed-length records only

4.3 VAX-11 RMS NETWORK ERROR REPORTING

For both local and remote file operations, VAX-11 RMS reports the success or failure of an operation by returning an RMS completion code. This primary status code is returned in both Register 0 (R0) and the completion status code field (STS) of the file access block (FAB) or record access block (RAB) specified in the original RMS service call. In addition, some RMS completion codes have secondary status information associated with them. This information is returned in the status value field (STV) of the FAB or RAB in the form of either another status code (usually from a different facility) or a value that qualifies the primary status code (such as a count). The RMS completion codes are listed in the *VAX-11 Record Management Services Reference Manual*.

When an RMS service call is issued for a network file operation, RMS enters into a dialog with the FAL server at the destination node, in order to perform the desired operation through the file system in use at that node. These cooperating processes communicate using the Data Access Protocol (DAP) to transfer data and to pass control and status information to each other. (The DECnet DIGITAL Network Architecture Data Access Protocol Functional Specification describes the DAP status codes.)

To promote network transparency, VAX-11 RMS attempts to map the status information returned by FAL into the RMS completion code that would be returned if the file had been accessed locally. It is not possible, however, to map every DAP status code directly into an RMS code applicable to local file access, particularly if the remote system does not use VAX-11 RMS to manage its files. Moreover, certain error conditions occur only in a network context. To handle these cases, several network-specific completion codes have been defined. These codes are summarized in Table 4-3.

Table 4-3: Network-Specific RMS Completion Codes

Status Code	Description
RMS$_ACS	error in access control string Indicates that the format of the access control string used in the file specification is invalid.
RMS$_BUG_DAP	Data Access Protocol error detected; DAP code = 'xxxxxxxx' Indicates that the operation failed because of a protocol error detected by either RMS or FAL. VAX-11 RMS returns this code in the STS field and a companion DAP code in the STV field. Note that a reproducible RMS$_BUG_DAP error indicates a DECnet software error condition that should be reported to DIGITAL in a Software Performance Report. However, a nonreproducible RMS$_BUG_DAP error, especially one that occurs on a communications line that has had RMS$_CRC errors reported, normally indicates a hardware malfunction.
RMS$_CRC	network DAP level CRC check failed Signals that file-level cyclic redundancy check (CRC) checksums computed by RMS and FAL did not match when compared at file close, thus indicating that the file is corrupted in some manner. This condition is caused usually by a hardware problem; if it occurs repeatedly, check the communications hardware. You should retry the file access. If DECnet event logging is enabled, you can set the Event Logger (EVL) to count CRC errors and display CRC error messages. For a connection between two VAX/VMS nodes, both RMS and FAL will independently log the event on their systems. (Event logging is described in the DECnet-VAX System Manager's Guide.)

(continued on next page)

Table 4-3 (Cont.): Network-Specific RMS Completion Codes

Status Code	Description
RMS$_CRE_STM	file was created in stream format Indicates that RMS has created the file in stream format with embedded carriage control because the format and carriage control specified in the RMS $CREATE call is not supported by the remote node.
RMS$_FTM	network file transfer mode precludes operation (SQO set) Indicates that RMS could not perform the requested operation because DAP file transfer mode was in effect. For network file access, setting the SQO bit in the FOP field of the FAB selects DAP file transfer mode. In this mode, the software blocks data records together for transmission over the network. This blocking increases data throughput and reduces CPU overhead. However, selection of this mode limits data transfers to one direction: either transmits using $PUT or $WRITE or receives using $GET or $READ. It also disallows other VAX-11 RMS record operations for the logical link until the record stream is terminated, via either $DISCONNECT or $CLOSE. Refer to Section 4.5.3 for additional information.
RMS$_NETFAIL	network operation failed at remote node Indicates that the requested operation could not be performed by the file system at the remote node. The STV field contains a FAL status code that describes the failure in the context of the remote system.
RMS$_NET	network operation failed at remote node; DAP code = 'xxxxxxxx' Indicates the same condition as the RMS$_NETFAIL code (see above) except that the accompanying DAP status code cannot be translated into a FAL status code. RMS$_NET is returned in the STS field and the DAP code in the STV field.
RMS$_NOD	error in node name Indicates that the node name portion of the file specification string has incorrect syntax.
RMS$_QUO	error in quoted string Indicates that the quoted string portion of the file specification (either the foreign-file-spec or task-spec string) has incorrect syntax.

(continued on next page)

Table 4-3 (Cont.): Network-Specific RMS Completion Codes

Status Code	Description
RMS$_SUPPORT	network operation not supported Indicates that VAX-11 RMS rejected the request because the operation requested is not supported over the network. The STV field contains either another RMS completion code or a FAL status code, depending on whether VAX-11 RMS at the local node or FAL at the remote node could not support the request.
RMS$_SUP	network operation not supported; DAP code = 'xxxxxxxx' Indicates the same condition as the RMS$_SUPPORT code except that the accompanying DAP status code cannot be translated into a FAL status code. RMS$_SUP is returned in the STS field and the DAP code in the STV field.

In general, RMS completion codes returned in response to network operations fall into one of the following categories:

1. The operation was successful.
2. The operation was not supported by the network.
3. The operation was attempted but failed.
4. An end-to-end file-level cyclic redundancy check (CRC) failed.
5. A DAP error was detected.

VAX-11 RMS reports status for each of these categories as follows:

1. Successful completion of a network operation is reported using the same RMS completion codes as those used to report the status of a local file operation. The RMS success completion code RMS$_NORMAL or an alternate success completion code (for example, RMS$_CRE_STM) is returned in the STS field. Depending on which RMS code is used, the STV field may or may not contain auxiliary information.

2. If an operation is not supported in a network context, the failure of the request is reported as either an RMS$_SUPPORT or an RMS$_SUP error. Most frequently returned is the RMS$_SUPPORT error, which has an associated secondary status code in the STV field. This secondary code is either another RMS completion code or a FAL status code, depending on whether RMS at the local node or FAL at the remote node could not support the request. The RMS$_SUP completion code is used only when RMS cannot map the DAP status code returned by FAL into a meaningful FAL status code. For RMS$_SUP, the uninterpreted DAP status code is returned in the STV field.

3. An operation supported over the network may fail while being processed by either the local or the remote file system. If the failure occurs at the local node, an appropriate RMS completion code is returned. (Examples of codes returned

when errors are detected locally by RMS are RMS$_ACS, RMS$_FTM, RMS$_NOD, and RMS$_QUO.) On the other hand, when a file operation fails while being processed by the file system at the remote node, RMS attempts to map the DAP status code generated by FAL into a corresponding RMS completion code. This mapping normally succeeds, but if an appropriate match cannot be found (because the error is specific to the remote file system), one of the following is returned to the user: RMS$_NETFAIL or RMS$_NET. Both of these codes indicate that the failure occurred at the remote system, but differ in the format of the contents of the STV field. If the DAP status code can be transformed into a FAL status code, RMS$_NETFAIL is returned with the FAL status code that describes the failure in terms of the remote file system. Otherwise, RMS$_NET is returned with the uninterpreted DAP status code in the STV field.

4. If a file-level CRC check failure occurs, the failure is signaled by means of the RMS$_CRC completion code for the $CLOSE service call. There is no secondary status information associated with this error condition. When a remote file is opened (or created) through VAX-11 RMS on a VAX/VMS node, RMS determines whether the remote FAL supports the DAP option of performing an end-to-end CRC check on the data accessed in the file. If FAL supports this option, then RMS and FAL agree to compute independently a cumulative CRC checksum based on the records (or blocks) each sends and/or receives. As part of the close operation, FAL compares the two checksums and reports status back to RMS. Thus an RMS$_CRC error alerts the user that the file has been corrupted during transfer. Repeated reports of this error for the same line indicate the possibility of a hardware failure.

5. During the exchange of DAP messages, should either node detect a protocol violation, RMS aborts the operation and returns the RMS$_BUG_DAP error code with a corresponding DAP status code in the STV field. A protocol violation is detected, for example, when a syntax check of a DAP message fails or RMS and FAL get out of synchronization (that is, a DAP message is received that is inappropriate for the current state).

4.4 HIGHER-LEVEL LANGUAGE REMOTE FILE ACCESS

Regardless of the programming language used, you access remote files in exactly the same way that you would access local files. To illustrate the way you access remote files, Example 4-1 provides a simple FORTRAN program to transfer a remote file on node TRNTO to the line printer on node BOSTON.

Example 4-1: FORTRAN Remote File Access Program

```
        PROGRAM TRANSFER.FOR
C
C       This program creates a sequential file with variable length
C       records from a sequential input file. The input and output
C       files are identified by the logical names SRC and DST,
C       respectively. For example, to print a file on a remote system
C       that resides on another system:
C
C       $ DEFINE SRC TRNTO::USER:[STOCKROOM.PAPER]INVENTORY.DAT
C       $ DEFINE DST BOSTON::LPA0:
C       $ RUN TRANSFER
C
        CHARACTER BUFFER*132
C
100     FORMAT (Q,A)
200     FORMAT (A)
C
C       Open the input and output files.
C
        OPEN (UNIT=1,NAME='SRC',TYPE='OLD',ACCESS='SEQUENTIAL',
     1       FORM='FORMATTED')
        OPEN (UNIT=2,NAME='DST',TYPE='NEW',ACCESS='SEQUENTIAL',
     1       FORM='FORMATTED',CARRIAGECONTROL='LIST',
     2       RECORDTYPE='VARIABLE')
C
C       Transfer records until end-of-file or other error condition.
C
10      READ (1,100,END=20,ERR=20) NCHAR,BUFFER(:NCHAR)
        WRITE (2,200) BUFFER(:NCHAR)
        GOTO 10
C
C       Close the input and output files.
C
20      CLOSE (UNIT=2)
        CLOSE (UNIT=1)
        END
```

The program in Example 4-1 uses standard I/O calls to transfer the file from one device to another. Initially, two DCL commands are used to define logical names for the files involved in the transfer. Note that you use the standard VAX/VMS file specification with a remote node name to specify the source and destination files. (If the remote node is other than VAX/VMS, the format of the file specification may differ.) The FORTRAN program must open the files. When opening a file, you must specify a unit over which the operation is to be performed. In this example, the access mode is sequential. Standard read and write calls serve to move the data from the source to the destination file. After all the records have been transferred, the program closes the channels, thereby terminating network operations. These operations are similar for applications in the other higher-level languages supported under VAX/VMS.

4.5 MACRO REMOTE FILE ACCESS

VAX/VMS provides a transparent programming interface for remote file access using VAX-11 MACRO. The following sections describe the use of VAX-11 RMS calls for VAX-11 MACRO remote file access, discuss network error mapping considerations, and present several examples of programs written in VAX-11 MACRO that illustrate the use of these calls.

4.5.1 Using VAX-11 RMS Service Calls

In general, you follow four steps when developing a MACRO remote file access application:

1. Initialize a set of data structures that VAX-11 RMS uses for processing RMS service requests (in the same way as for local MACRO program development)
2. Open the files for processing and establish a record stream for each file
3. Issue the appropriate RMS service calls to perform the actual file-handling operations
4. Disconnect the record streams and close the files after file processing has been completed

4.5.1.1 File Access Blocks (FABs) and Record Access Blocks (RABs) - These serve as the discrete data structures that VAX-11 RMS uses to process service requests. In general, you should allocate one FAB for every file your program will open and one RAB per file to establish a record stream. The parameters associated with these control blocks define the exact characteristics of the files designated for processing. One of these parameters is the network file specification. Through the use of either a logical name or a complete file specification string, you designate the file that you want to access.

4.5.1.2 $OPEN and $CONNECT - Once you have defined the user control blocks, you can then open the files and establish a record stream to each one. Use the RMS service calls $OPEN and $CONNECT in conjunction with the FAB and RAB to prepare for file processing. When you issue the $OPEN call, you effectively establish a logical link connection with the remote FAL.

When the file is open, you can perform file and record operations through the use of standard RMS service calls such as $GET, $PUT, and $UPDATE. For added flexibility, VAX/VMS supports block I/O RMS calls for writing and reading blocks of data to and from remote files.

4.5.1.3 $DISCONNECT and $CLOSE - Finally, after processing completes, you should disconnect all record streams and close all files. First, use the $DISCONNECT call to disconnect the RAB, and then use the $CLOSE call to close the FAB, thereby signaling the completion of network operations. This procedure terminates the logical link in an orderly fashion. Note that the $CLOSE macroinstruction will automatically disconnect the record stream for the RAB associated with the specified FAB.

4.5.2 VAX-11 RMS Service Call Summary

Table 4-4 summarizes the VAX-11 RMS service calls you can use for MACRO remote access operations: file processing, record processing, block I/O processing, and file specification processing. This table lists calls supported for file operations between two VAX/VMS nodes over the network. The following operations are not supported in this environment: $ENTER, $NXTVOL, $REMOVE, and $RENAME. (If one of these operations is requested by a VAX/VMS node communicating via DECnet with another VAX/VMS node, the RMS$_SUPPORT (network operation not supported) error message is returned.)

Appendix B describes network considerations relating to the use of VAX-11 RMS control blocks associated with the calls listed in Table 4-4. For a complete explanation of the syntax and parameters for each call, refer to the VAX-11 Record Management Services Reference Manual.

Table 4-4: VAX-11 RMS Service Calls for Run-time Remote File Access

Name of MACRO	Description of Service
File Processing	
$CLOSE	Closes an open file, optionally providing for file disposition, and terminates file processing
$CREATE	Creates a new file with the attributes specified, opens it, and initiates file processing
$DISPLAY	Returns the attributes of a file to the user program
$ERASE	Deletes a closed file and removes its directory entry
$EXTEND	Increases space allocated to an open disk file
$OPEN	Opens an existing file, optionally returns its file attributes, and initiates file processing
Record Processing	
$CONNECT	Establishes a record stream by associating a RAB with an open file
$DELETE	Deletes a record from a relative or indexed file
$DISCONNECT	Terminates a record stream by disconnecting a RAB from an open file
$FIND	Locates and positions to the specified record in the file
$FLUSH	Forces buffered records and modified file attributes to be written to a file

(continued on next page)

Table 4-4 (Cont.): VAX-11 RMS Service Calls for Run-time Remote File Access

Name of MACRO	Description of Service
$FREE	Unlocks all records previously locked by the record stream
$GET	Retrieves a record from a file
$PUT	Writes a record to a file
$RELEASE	Unlocks a record specified by its record file address
$REWIND	Positions to the first record or block of a file
$TRUNCATE	Truncates a sequential file
$UPDATE	Modifies a record in a file
$WAIT	Awaits the completion of an asynchronous record operation
Block I/O Processing	
$READ	Reads data in block I/O mode
$SPACE	Positions forward or backward in a file to a block boundary
$WRITE	Writes data in block I/O mode
File Specification Processing	
$PARSE	Parses a file specification
$SEARCH	Searches a directory for the next file name that matches the file specification template provided

4.5.3 VAX-11 RMS Programming Notes and Restrictions

This section discusses programming topics related to the network of interest primarily to the MACRO programmer. Restrictions on the use of the VAX-11 RMS interface in a network environment are also presented. Refer to Appendix B for additional information on the use of RMS control blocks.

4.5.3.1 Name Block - The DID, DVI, and FID fields of the Name Block ($NAM) are not supported for remote file access; these fields are not used as input and are zeroed on output. If you set the NAM bit in the file-processing options (FOP) field of the FAB to request "open by NAM block," this option will be ignored and the open will proceed based on the other fields of the $FAB and $NAM blocks.

4.5.3.2 File Specification Processing - Using the RMS $PARSE service call to parse a file specification that contains a node name does not incur any additional overhead in a network context. VAX-11 RMS parses/merges the primary, default, and related file specification strings into an expanded name string without invoking a remote FAL server. Because the file parse is performed locally, the following control block fields are affected on output:

- In the expanded name string (ESA/ESL) of the Name Block, a logical device name is returned without being translated if it was encountered after a node name was found during the file parse operation, and process default device and directory name strings are not applied.
- In the file name status field (FNB) of the Name Block, status bits are not valid if the quoted string format of a file specification is used.
- The device characteristics fields (DEV and SDC) of the FAB are not updated to reflect the actual characteristics of the remote device.

In contrast to $PARSE, execution of the RMS service call $SEARCH will cause a logical link to be created to communicate with a remote FAL server to perform the directory search function. Repeated calls to $SEARCH using the same FAB, however, will cause RMS to use the same logical link to FAL until the search sequence is complete. Furthermore, execution of an $OPEN, $CREATE, or $ERASE call following a $SEARCH call will cause VAX-11 RMS to establish a new logical link with FAL to perform the file access operation.

4.5.3.3 FOP File Disposition Options on Close - Three options of the FOP field of the FAB that affect the disposition of the remote file on close require that you supply a resultant name string via the Name Block ($NAM) as input to the $CLOSE service. The options are:

- DLT to delete the file at the remote node.
- SCF to submit the command file for execution at the remote node. You may also specify that the file be deleted after execution by setting both the SCF and DLT bits.
- SPL to print one copy of the file at the remote node. You may also specify that the file be deleted after it is printed by setting both the SPL and DLT bits.

These options may be requested at either open or close time, but they are performed as part of the close operation. The resultant string must be the same one received by specifying the Name Block as input to the $CREATE or $OPEN calls. If you specified one or more of the FOP options previously discussed and you receive a VAX-11 RMS completion code of RMS$_SUPPORT (network operation not supported) on $CLOSE, it means that the file was closed at the (non-VAX/VMS) target node, but none of the options were performed.

4.5.3.4 FOP Option for Increasing File Transfer Throughput - The Data Access Protocol used by VAX-11 RMS and its partner FAL defines two major file access modes: DAP file transfer mode (FTM) and DAP record access mode (RAM). VAX-11 RMS must translate each user-specified access method (for example, sequential, random, block I/O) into one of these DAP access modes in order to perform remote file access operations. VAX/VMS supports both DAP file access modes, whereas most DECnet implementations other than VAX/VMS support only FTM.

FTM is designed to speed up the most common network operation: copying files. FTM allows VAX-11 RMS to exchange fewer DAP messages with the remote FAL than does RAM when transferring a file. FTM also permits data records to be blocked together for transmission, thereby potentially reducing the number of QIO system service calls required to move the data. The combined effect is that data throughput increases as the average record size of the file decreases. A side benefit is that this lessens operating system overhead as compared with RAM.

Although FTM offers efficiency, it is restrictive. It requires that the file be accessed sequentially (by records or blocks) and that the data be moved in one direction (either sent to or retrieved from the file). Thus once the record stream is established via a $CONNECT call, only $GET/$READ or $PUT/$WRITE requests are permitted until the record stream is terminated via a $DISCONNECT or a $CLOSE call. If you attempt to mix $GET/$PUT or $READ/$WRITE requests or to issue other RMS service calls (such as $DISPLAY, $EXTEND, $FIND, or $UPDATE), VAX-11 RMS will return an RMS$_FTM error.

In contrast, RAM is designed to provide a wide range of functions. RAM supports both sequential and random access (by records or blocks), numerous record operations such as $FREE and $UPDATE, and dynamic switching of access modes. RAM is also useful for performing intermixed $GET/$PUT operations to a bidirectional unit record device such as a remote terminal or a remote task. However, RAM offers flexibility at the expense of efficiency. Each VAX-11 RMS service call results in an exchange of DAP messages with the remote FAL. Data messages cannot be blocked with each other (even when the file is accessed sequentially) because RAM requires that FAL acknowledge the completion of each file operation before the next one can be requested.

When you open or create a remote file, VAX-11 RMS examines the SQO (sequential-only) bit in the FOP field of the FAB to determine whether to enter FTM or RAM. If the SQO bit is set, VAX-11 RMS selects FTM, and you are limited to reading or writing data in a sequential manner. If the SQO bit is not set, RMS selects RAM unless the remote FAL supports only FTM. If this is the case, then RMS overrides your SQO request and enters FTM. Consequently, if you write a program in VAX-11 MACRO either to open a remote file and sequentially read data or to create a remote file and sequentially write data, it is recommended that you select the sequential-only FOP option. This option will improve data throughput for transfers between VAX/VMS nodes. Where the remote partner is other than VAX/VMS, the SQO option will have an effect only if the remote node supports both FTM and RAM.

4.5.3.5 File Sharing - File sharing between VAX/VMS nodes is supported over the network. Different programs can open the same remote file for shared access through use of the file sharing (FAB$B_SHR) field of the FAB. Within a single program, however, only one record stream can be active for each open remote file. You cannot set the FAB$V_MSE (multistream access enabled) bit of the FAB$B_SHR field and then issue multiple $CONNECT calls for the same file. You may, however, reestablish a record stream by issuing a $CONNECT after a disconnect operation has been performed without closing and reopening the file.

4.5.3.6 Restriction on Access to Files on Magnetic Tape - DECnet-VAX does not support access to magnetic tape files located on a remote VAX/VMS system. This restriction exists because FAL is not able to mount a magnetic tape volume in order to gain access to it.

4.5.3.7 Task-to-Task Communication - When writing a program to communicate with a remote task, you may treat that remote task as though it were a unit record device, having characteristics similar to those of a VAX/VMS mailbox. The information returned in the device characteristics field (DEV) of the FAB block reflects this orientation. As a result, sequential $GET/$PUT requests are appropriate for exchanging data with a remote task, whereas random access requests are not.

The Data Access Protocol is not used for task-to-task communication. Consequently, the sequential-only (SQO) option in the FOP field of the FAB has no effect on data throughput. Each $GET/$PUT request results in a QIO system service request to receive or transmit data. Furthermore, if one task breaks the logical link, its partner task will receive an RMS$_EOF (end-of-file) error in response to an outstanding $GET request. Breaking the link is analogous to pressing CTRL/Z on a terminal to indicate an end-of-file condition. In response to an outstanding $PUT request, the partner task will receive an RMS$_SYS (system QIO directive) error.

4.5.4 MACRO Programming Examples

The following examples illustrate the use of VAX-11 RMS service calls in MACRO programs that process files on remote nodes. For a general discussion of the use of VAX-11 RMS at the local node, refer to the VAX-11 Record Management Services Reference Manual.

4.5.4.1 MACRO Remote File Transfer Example - Example 4-2 illustrates the VAX-11 MACRO counterpart to the VAX-11 FORTRAN remote file transfer program in Example 4-1.

Example 4-2: RMS File Transfer Program

```
        .TITLE  CREATESEQ - CREATE SEQUENTIAL FILE
        .IDENT  /V001/
;
; This program creates a sequential file with variable length records from
; a sequential input file. The input and output files are identified by the
; logical names SRC and DST, respectively. For example, to create a file on
; one node from a file residing on another node:
;
;       $ DEFINE SRC TRNTO::USER:[STOCKROOM.PAPER]INVENTORY.DAT
;       $ DEFINE DST BOSTON::TEMP:[ARCHIVE]INVENTORY.DAT
;       $ RUN CREATESEQ
;

;***************************************************************************

        .SBTTL  Control block and buffer storage
        .PSECT  DATA            NOEXE,LONG
;
; Define the source file FAB and RAB control blocks.
;
SRC_FAB:
        $FAB    FAC=<GET>,-                 ; File access for GET only
                FOP=<SQO>,-                 ; Request DAP file transfer mode
                FNM=<SRC>                   ; Name of input file
SRC_RAB:
        $RAB    FAB=SRC_FAB,-               ; Address of associated FAB
                RAC=SEQ,-                   ; Sequential record access
                UBF=BUFFER,-                ; Buffer address
                USZ=BUFFER_SIZE             ; Buffer size
;
; Define the destination file FAB and RAB control blocks.
;
DST_FAB:
        $FAB    FAC=<PUT>,-                 ; File access for PUT only
                FOP=<SQO>,-                 ; Request DAP file transfer mode
                FNM=<DST>,-                 ; Name of output file
                ORG=SEQ,-                   ; Sequential file organization
                RFM=VAR,-                   ; Variable record format
                RAT=<CR>                    ; Implied carriage control
DST_RAB:
        $RAB    FAB=DST_FAB,-               ; Address of associated FAB
                RAC=SEQ,-                   ; Sequential record access
                RBF=BUFFER                  ; Buffer address
;
; Allocate buffer to be the size of the largest record that will be read.
;
BUFFER: .BLKB   132                         ; Buffer for input and output
        BUFFER_SIZE=.-BUFFER                ; Buffer size

;***************************************************************************

        .SBTTL  Mainline
        .PSECT  CODE            NOWRT,BYTE
;
; Start of program; put FAB and RAB addresses in registers for efficiency.
;
        .ENTRY  CREATESEQ,^M<>              ; Entry point
        MOVAB   W^SRC_FAB,R6                ; Get address of input FAB
        MOVAB   W^SRC_RAB,R7                ; Get address of input RAB
        MOVAB   W^DST_FAB,R8                ; Get address of output FAB
        MOVAB   W^DST_RAB,R9                ; Get address of output RAB
```

```
;
; Open the source and destination files.
;
        $OPEN    FAB=R6                   ; Open input file
        BLBC     R0,EXIT                  ; Branch on failure
        $CONNECT RAB=R7                   ; Connect input record stream
        BLBC     R0,EXIT                  ; Branch on failure
        $CREATE FAB=R8                    ; Create output file
        BLBC     R0,EXIT                  ; Branch on failure
        $CONNECT RAB=R9                   ; Connect output record stream
        BLBC     R0,EXIT                  ; Branch on failure
;
; Transfer records until end-of-file is encountered.
;
LOOP:   $GET     RAB=R7                   ; Read next record from input file
        BLBC     R0,ERROR                 ; Branch on failure
        MOVW     RAB$W_RSZ(R7),-          ; Copy length of record just
                 RAB$W_RSZ(R9)            ;  read to output RAB
        $PUT     RAB=R9                   ; Write record to output file
        BLBC     R0,ERROR                 ; Branch on failure
        BRB      LOOP                     ; Process next record
ERROR:  CMPL     R0,#RMS$_EOF             ; Was it an end-of-file?
        BNEQ     EXIT                     ; Branch if not
;
; Close the source and destination files.
;
; Note that a $DISCONNECT call is not required prior to closing a file
; because the $CLOSE call performs an implied disconnect of the record stream.
;
        $CLOSE   FAB=R8                   ; Close output file
        BLBC     R0,EXIT                  ; Branch on failure
        $CLOSE   FAB=R6                   ; Close input file
;
; Exit to VMS with RMS completion code in R0 to signal success or
; failure of program execution.
;
EXIT:   $EXIT_S R0                        ; Exit with RMS completion code in R0
        .END     CREATESEQ                ; Specify starting address of program
```

4.5.4.2 VAX-11 MACRO Remote File Spooling Example - Example 4-3 spools a sequential file to the printer on the remote node.

Example 4-3: RMS Spool File Program

```
        .TITLE  SPOOL - PRINT AND DELETE FILE
        .TITLE  /V001/
;
; This program spools one copy of a file specified by the logical name
; DST to the printer at the remote node where it resides. After being
; printed, the file is deleted. For example:
;
;       $ DEFINE DST 'BOSTON''DEMO NETWORK''::[.TEST]PRINTME.DAT'
;       $ RUN SPOOL
;

;*******************************************************************

        .SBTTL  Control block and buffer storage
        .PSECT  DATA            NOEXE,LONG
;
; Define the destination file FAB and NAM control blocks.
;
DST_FAB:
        $FAB    FOP=<SPL,DLT>,-             ; Print and delete file
                FNM=<DST>,-                 ; Name of file
                NAM=DST_NAMBLK              ; Name block address
DST_NAMBLK:
        $NAM    RSA=DST_RS,-                ; Resultant name string buffer address
                RSS=NAM$C_MAXRSS            ; Resultant name string buffer size
;
; Allocate buffer for resultant name string.
;
DST_RS:         .BLKB   NAM$C_MAXRSS        ; Largest resultant name string

;*******************************************************************

        .SBTTL  Mainline
        .PSECT  CODE            NOWRT,BYTE
;
; Start of program
;
        .ENTRY  SPOOL,^M<>                  ; Entry point
;
; Open and close the file with the FOP print and delete options specified.
;
        $OPEN   FAB=DST_FAB                 ; Open the file
        BLBC    R0,EXIT                     ; Branch on failure
        $CLOSE  FAB=DST_FAB                 ; Close the file which will cause it
                                            ; to be printed and deleted
;
; Exit to VMS with RMS completion code in R0 to signal success or
; failure of program execution.
;
EXIT:   $EXIT_S R0                          ; Exit with RMS completion code in R0
        .END    SPOOL                       ; Specify starting address of program
```

4.5.4.3 VAX-11 MACRO Remote File Random Access Example - Example 4-4 accesses a relative file randomly and transfers selected records from one device to another.

Example 4-4: RMS Random Access Program

```
        .TITLE  RANDOM - RANDOM ACCESS EXAMPLE
        .IDENT  /V001/
;
; This program accesses a relative file randomly by relative record
; number. It creates a sequential output file from selected records
; from the input file. The input and output files are specified by
; the logical names SRC and DST, respectively. For example:
;
;       $ DEFINE SRC TRNTO::USER:[P3602.DAT]EVENTLOG.DAT
;       $ DEFINE DST BOSTON::TEMP.LIS
;       $ RUN RANDOM
;

;*********************************************************************

        .SBTTL  Control block and buffer storage
        .PSECT  DATA            NOEXE,LONG
;
; Define the source file FAB and RAB control blocks.
;
SRC_FAB:
        $FAB    FAC=<GET>,-                 ; File access for GET only
                FNM=<SRC>                   ; Name of input file
SRC_RAB:
        $RAB    FAB=SRC_FAB,-               ; Address of associated FAB
                RAC=KEY,-                   ; Access by relative record number
                KBF=KEY_BUFFER,-            ; Key buffer address
                KSZ=4,-                     ; Key size
                UBF=BUFFER,-                ; Buffer address
                USZ=BUFFER_SIZE             ; Buffer size
;
; Define the destination file FAB and RAB control blocks.
;
DST_FAB:
        $FAB    FAC=<PUT>,-                 ; File access for PUT only
                FOP=<SQO>,-                 ; Request DAP file transfer mode
                FNM=<DST>,-                 ; Name of output file
                ORG=SEQ,-                   ; Sequential file organization
                RAT=<CR>                    ; Implied carriage control
DST_RAB:
        $RAB    FAB=DST_FAB,-               ; Address of associated FAB
                RAC=SEQ,-                   ; Sequential record access
                RBF=BUFFER                  ; Buffer address
;
; Allocate buffer to be the size of the largest record that will be read.
;
BUFFER: .BLKB   512                         ; Buffer for input and output
        BUFFER_SIZE=.-BUFFER                ; Buffer size
;
; Specify record number list and allocate key buffer.
;
RECORD_LIST:
        .LONG   4,1,10,0                    ; Record selection list terminated by 0
KEY_BUFFER:
        .LONG   0                           ; Buffer to store relative record numbe

;*********************************************************************
```

```
        .SBTTL  Mainline
        .PSECT  CODE            NOWRT,BYTE
;
; Start of program; put FAB and RAB addresses in registers for efficiency.
;
        .ENTRY  RANDOM,^M<>             ; Entry point
        MOVAB   W^SRC_FAB,R6            ; Get address of input FAB
        MOVAB   W^SRC_RAB,R7            ; Get address of input RAB
        MOVAB   W^DST_FAB,R8            ; Get address of output FAB
        MOVAB   W^DST_RAB,R9            ; Get address of output RAB
;
; Open the source and destination files.
;
        $OPEN   FAB=R6                  ; Open input file
        BLBC    R0,EXIT                 ; Branch on failure
        MOVB    FAB$B_RFM(R6),-         ; Copy record format attribute
                FAB$B_RFM(R8)           ;  to output FAB
        MOVW    FAB$W_MRS(R6),-         ; Copy maximum record size
                FAB$W_MRS(R8)           ;  attribute to output FAB
        $CONNECT RAB=R7                 ; Connect input record stream
        BLBC    R0,EXIT                 ; Branch on failure
        $CREATE FAB=R8                  ; Create output file
        BLBC    R0,EXIT                 ; Branch on failure
        $CONNECT RAB=R9                 ; Connect output record stream
        BLBC    R0,EXIT                 ; Branch on failure
;
; Read each record specified in list and write it to the destination file.
;
        MOVAB   W^RECORD_LIST,R5        ; Get record number vector
LOOP:   MOVL    (R5)+,W^KEY_BUFFER      ; Get next record number
        BEQL    CLOSE                   ; All done if list terminator found
        $GET    RAB=R7                  ; Read specified record from input file
        BLBC    R0,EXIT                 ; Branch on failure
        MOVW    RAB$W_RSZ(R7),-         ; Copy length of record just
                RAB$W_RSZ(R9)           ;  read to output RAB
        $PUT    RAB=R9                  ; Write record to output file
        BLBC    R0,EXIT                 ; Branch on failure
        BRB     LOOP                    ; Process next record
;
; Close the source and destination files.
;
CLOSE:  $CLOSE  FAB=R8                  ; Close output file
        BLBC    R0,EXIT                 ; Branch on failure
        $CLOSE  FAB=R6                  ; Close input file
;
; Exit to VMS with RMS completion code in R0 to signal success or
; failure of program execution.
;
EXIT:   $EXIT_S R0                      ; Exit with RMS completion code in R0
        .END    RANDOM                  ; Specify starting address of program
```

4.5.4.4 VAX-11 MACRO Remote File Indexed Access Example - Example 4-5 creates an indexed file with three keys from a sequential file on the local node.

Example 4-5: RMS Indexed File Program

```
        .TITLE  CREATEIDX - CREATE INDEXED FILE
        .IDENT  /V001/
;
; This program creates an indexed file with three keys from a
; sequential file containing a name and address list. The record
; format of the input file is shown below:
;
;       First Name       Column  01-10
;       Middle Initial   Column  11-11
;       Last Name        Column  12-26
;       Street           Column  27-46
;       City             Column  47-58
;       State            Column  59-60
;       Zip Code         Column  61-65
;
; The input and output files are specified by the logical names SRC
; and DST, respectively. For example:
;
;       $ DEFINE SRC BOSTON::DBB1:[TEST]INPUT.DAT
;       $ DEFINE DST TRNTO::DRA4:[RMS.FILES]OUTPUT.DAT
;       $ RUN CREATEIDX
;
;*********************************************************************

        .SBTTL  Control block and buffer storage
        .PSECT  DATA            NOEXE,LONG
;
; Define the source file FAB and RAB control blocks.
;
SRC_FAB:
        $FAB    FAC=<GET>,-                 ; File access for GET only
                FOP=<SQO>,-                 ; Request DAP file transfer mode
                FNM=<SRC>                   ; Name of input file
SRC_RAB:
        $RAB    FAB=SRC_FAB,-               ; Address of associated FAB
                RAC=SEQ,-                   ; Sequential record access
                UBF=BUFFER,-                ; Buffer address
                USZ=BUFFER_SIZE             ; Buffer size
;
; Define the destination file FAB and RAB control blocks.
;
DST_FAB:
        $FAB    FAC=<PUT>,-                 ; File access for PUT only
                FNM=<DST>,-                 ; Name of output file
                ORG=IDX,-                   ; Indexed file organization
                RFM=FIX,-                   ; Fixed length records
                RAT=<CR>,-                  ; Implied carriage control
                MRS=BUFFER_SIZE,-           ; Record size
                BKS=1,-                     ; Bucket size
                XAB=DST_KEY0                ; Address of start of XAB chain
DST_RAB:
        $RAB    FAB=DST_FAB,-               ; Address of associated FAB
                RAC=SEQ,-                   ; Sequential record access
                RBF=BUFFER,-                ; Buffer address
                RSZ=BUFFER_SIZE             ; Buffer size
;
; Define Key Definition XABs to describe the three keys.
;
```

```
DST_KEY0:                                      ; Primary key is State
        $XABKEY REF=0,-                        ; Key reference number
                POS=58,-                       ; Starting key position
                SIZ=2,-                        ; Key size
                FLG=<DUP>,-                    ; Duplicate keys are allowed
                NXT=DST_KEY1                   ; Address of next XAB in chain
DST_KEY1:                                      ; 1st alternate key is City
        $XABKEY REF=1,-                        ; Key reference number
                POS=46,-                       ; Starting key position
                SIZ=12,-                       ; Key size
                FLG=<DUP>,-                    ; Duplicate keys are allowed
                NXT=DST_KEY2                   ; Address of next XAB in chain
DST_KEY2:                                      ; 2nd alternate key is Name
        $XABKEY REF=2,-                        ; Key reference number
                POS=<11,0,10>,-                ; Segmented key consisting of Last Name,
                SIZ=<15,1,1>,-                 ; First Initial, and Middle Initial
                NXT=0                          ; Designate end of XAB chain
;
; Allocate buffer to be the size of the largest record that will be read.
;
BUFFER: .BLKB   65                             ; Buffer for input and output
        BUFFER_SIZE=.-BUFFER                   ; Buffer size

;**********************************************************************

        .SBTTL  Mainline
        .PSECT  CODE            NOWRT,BYTE
;
; Start of program
;
        .ENTRY  CREATEIDX,^M<>                 ; Entry point
;
; Open the source and destination files.
;
        $OPEN   FAB=SRC_FAB                    ; Open input file
        BLBC    R0,EXIT                        ; Branch on failure
        $CONNECT RAB=SRC_RAB                   ; Connect input record stream
        BLBC    R0,EXIT                        ; Branch on failure
        $CREATE FAB=DST_FAB                    ; Create output file
        BLBC    R0,EXIT                        ; Branch on failure
        $CONNECT RAB=DST_RAB                   ; Connect output record stream
        BLBC    R0,EXIT                        ; Branch on failure
;
; Transfer records until end-of-file is reached.
;
LOOP:   $GET    RAB=SRC_RAB                    ; Read next record from input file
        BLBC    R0,ERROR                       ; Branch on failure
        $PUT    RAB=DST_RAB                    ; Write record to output file
        BLBC    R0,ERROR                       ; Branch on failure
        BRB     LOOP                           ; Process next record
ERROR:  CMPL    R0,#RMS$_EOF                   ; Was it an end-of-file?
        BNEQ    EXIT                           ; Branch if not
;
; Close the source and destination files.
;
        $CLOSE  FAB=DST_FAB                    ; Close output file
        BLBC    R0,EXIT                        ; Branch on failure
        $CLOSE  FAB=SRC_FAB                    ; Close input file
;
; Exit to VMS with RMS completion code in R0 to signal success or
; failure of program execution.
;
EXIT:   $EXIT_S R0                             ; Exit with RMS completion code in R0
        .END    CREATEIDX                      ; Specify starting address of program
```

CHAPTER 5

TASK-TO-TASK COMMUNICATION

Task-to-task communication is a feature common to all DECnet implementations that allows two programs, running under the same or different operating systems, to communicate with each other regardless of the programming languages used. For example, a FORTRAN program running on the VAX/VMS system at node BOSTON of our network example could exchange messages with a MACRO program running on the RSX-11M system at node DALLAS. The fact that these programs use different programming languages and run under different operating systems is of no concern to the DECnet software, which translates system-dependent language calls into a common set of network protocol messages. (Note that in the context of task-to-task communication, the terms "task" and "program" are used synonymously.)

DECnet-VAX supports two forms of task-to-task communication: transparent and nontransparent. Transparent communication provides all the basic functions necessary for a program in VAX-11 MACRO or a higher-level language to communicate with other programs over the network. Nontransparent communication allows the programmer to use more network specific functions.

A simple analogy differentiates these two forms of communication. Transparent communication is analogous to device-independent I/O under VAX/VMS. This form of I/O lets you move data with little concern for the way this is accomplished. Likewise, transparent communication allows you to simply move data across the network without necessarily knowing that you are using DECnet software. Nontransparent communication, on the other hand, is analogous to device-dependent I/O wherein you are interested in specific characteristics of the device that you want to access. A nontransparent task, in turn, can use network-specific features to monitor the communication process.

This chapter defines the forms of DECnet-VAX intertask communication and the general procedures necessary to implement them. Particular attention is paid to the DECnet interface with higher-level languages. The information and examples presented herein provide the necessary framework for the discussion of task-to-task communication in each higher-level language user guide. The programmer should also be familiar with this material before reading Chapters 6 and 7, which provide examples of transparent and nontransparent communication using system services.

5.1 TRANSPARENT COMMUNICATION

Transparent communication provides the basic functions necessary for a task to communicate with another task over the network. These functions include the initiation and completion of a logical link connection, the orderly exchange of messages between both tasks, and the controlled termination of the communication process. To implement these functions, you can program your transparent task in any of the higher-level languages supported over the network or in VAX-11 MACRO, using RMS service calls or system service calls.

One way to view transparent communication is to look at the programming required to develop such an application. Transparent access provides the minimum functions necessary to communicate over the network using standard I/O operations. When accessing the network transparently, you use no DECnet-specific calls to perform these functions; rather, you use standard RMS I/O service calls or the normal I/O statements provided by the applicable higher-level language to access a sequential record-oriented device. (The remote task is modeled as a VAX/VMS mailbox to which you can perform read and write operations.) You can also use $QIO system service calls to perform transparent communication, as described in Chapter 6.

Example 5-1 illustrates a simple example of VAX-11 FORTRAN transparent communication. In the source FORTRAN task that initiates the logical link request, you use a standard open call to specify the remote task to which you want to connect. In turn, the remote task issues an open call to complete the logical link connection. A coordinated set of read and write operations enable the exchange of information over the link. To terminate the connection, the source task executes a close call to break the logical link. When the remote task receives this close call, it issues a close call which completes the link termination process. The remaining sections of this chapter discuss the calls that you would use to develop such an application.

Example 5-1: FORTRAN Task-to-Task Communication

```
        PROGRAM TEST3.FOR
C
C       This program prompts the user for the part number of an item
C       in inventory and responds with the number of units in stock.
C       The information is obtained by communicating with a program
C       (TEST4) on another node that has access to the inventory data.
C
C       Before running this program, the logical name TASK must be
C       defined to refer to the target program. For example:
C
C       $ DEFINE TASK 'TRNTO::''TASK=TEST4'''
C       $ RUN TEST3
C
        CHARACTER PARTNO*5
        INTEGER PARTCOUNT
C
100     FORMAT (/,'$Enter part number: ')
200     FORMAT (A)
300     FORMAT (I4)
400     FORMAT ('0Inventory for part number ',A,' is: ',I4)
C
C       Establish a logical link with the target task.
C
    ❶   OPEN (UNIT=1,NAME='TASK',ACCESS='SEQUENTIAL',
     1       FORM='FORMATTED',CARRIAGECONTROL='NONE',TYPE='NEW')
C
C       Prompt the user for a part number, send it to the target task,
C       read reply of quantity on hand, and finally display the answer
C       for the user. Repeat the cycle until the user enters 'EXIT' for
C       a part number.
C
```

```
10        TYPE 100
          ACCEPT 200, PARTNO
          IF (PARTNO .EQ. 'EXIT') GOTO 20
        ❹ WRITE (1,200) PARTNO
          READ (1,300) PARTCOUNT
          TYPE 400, PARTNO, PARTCOUNT
          GOTO 10
C
C         Disconnect the logical link.
C
20      ❺ CLOSE (UNIT=1)
          END
```

```
  $ !     TEST4.COM
  $ !
  $ !     This command procedure executes the program TEST4 after
  $ !     being started by a task-to-task connection request issued
  $ !     by TEST3.
  $ !
❷ $ RUN SYS$LOGIN:TEST4.EXE
  $ !
  $ !     Purge old log files generated by this command procedure.
  $ !
  $ PURGE/KEEP=2 SYS$LOGIN:TEST4.LOG
  $ EXIT
```

```
          PROGRAM TEST4.FOR
C
C         This is the target program that communicates with TEST3.
C         For each part number received from the source task, the
C         number of units in stock is determined, and this value is
C         returned.
C
C         To complete the logical link with its initiator, this program
C         uses the logical name SYS$NET as the file specification in an
C         open statement.
C
          CHARACTER PARTNO*5
          INTEGER PARTCOUNT
C
100       FORMAT (A)
200       FORMAT (I4)
C
C         Complete the logical link connection.
C
       ❸  OPEN (UNIT=1,NAME='SYS$NET',ACCESS='SEQUENTIAL',
       1       FORM='FORMATTED',CARRIAGECONTROL='NONE',TYPE='OLD')
C
C         Process requests until end-of-file is reached. (This is the
C         error condition returned when the source task breaks the
C         logical link connection.)
C
10       ❹ READ    (1,100,END=20) PARTNO
C
C         Perform appropriate processing to obtain the part count value
C         and transmit this back to the source task.
C
          CALL INSTOCK (PARTNO,PARTCOUNT)
        ❹ WRITE (1,200) PARTCOUNT
          GOTO 10
```

```
C
C       Disconnect the logical link.
C
20    ❺ CLOSE    (UNIT=1)
        END
```

Notes on Example 5-1:

❶ The source task, TEST3, requests a logical link connection to the target task, TEST4.

❷ When the remote node receives a connection request, the command procedure TEST4.COM is executed. This command procedure must reside under the default directory associated with the account being accessed. TEST4.COM contains a RUN statement that causes the program TEST4.EXE to be executed.

❸ TEST4 completes the logical link connection via SYS$NET. Note that the unit numbers in the source and target programs need not be the same.

❹ TEST3 and TEST4 send and receive data messages.

❺ TEST3 disconnects the logical link and thereby terminates the communication process.

Because DECnet-VAX translates system-dependent language calls into the same set of messages that permit task-to-task communication, any task programmed in VAX-11 MACRO or one of the higher-level languages can communicate with a remote task programmed in the same or a different language. More specifically, for communication between tasks that run on VAX/VMS nodes, the language in which you access the network has no effect on the communication process. The VAX-11 FORTRAN source task in Example 5-1 could just as easily communicate with a MACRO task on node TRNTO.

5.2 NONTRANSPARENT COMMUNICATION

Nontransparent communication provides the same basic functions as transparent communication plus additional system service and I/O functions supported by DECnet-VAX (as described in Chapter 7.) In particular, a nontransparent task can create and use a VAX/VMS mailbox to receive information that would otherwise remain inaccessible to a transparent task. Thus you can use optional network protocol features such as optional user data on connects and disconnects, and **interrupt messages**. Also, certain nontransparent tasks that have a mailbox can receive and process multiple inbound connection requests. The discussion that follows highlights the distinctions between the two types of access to emphasize the additional capabilities that nontransparent access provides.

Transparent communication offers the minimum functions necessary for initiating and completing a logical link connection, exchanging messages, and terminating the logical link. In fact, these functions are actually a subset of a larger group of functions defined for nontransparent communication. The entire set of functions are as follows:

- **Initiating a logical link connection**
  - Requesting a logical link to a remote task[1]
  - Declaring a network name and processing multiple connection requests
- **Completing a logical link connection**
  - Rejecting a logical link connection request
  - Accepting a logical link connection request[1]
- **Exchanging messages**
  - Sending and receiving data messages[1]
  - Sending and receiving interrupt messages
- **Terminating a logical link**
  - Synchronously disconnecting the logical link
  - Aborting the logical link[1]

Nontransparent tasks can use any or all of these functions to extend the basic capabilities offered under transparent communication.

5.2.1 Mailboxes and Mailbox Messages

In general, nontransparent tasks can use a mailbox to receive information about particular network operations. Mailbox messages are of three types:

- Messages that result from the use of certain system service calls (including optional user data)
- Interrupt messages
- Network status messages

Nontransparent functions that indirectly cause mailbox messages to be placed in the receiver's mailbox include calls for initiating and completing logical links, and calls for terminating links. Figure 5-1 illustrates the use of mailboxes by nontransparent tasks.

1. Minimum subset for transparent task-to-task communication.

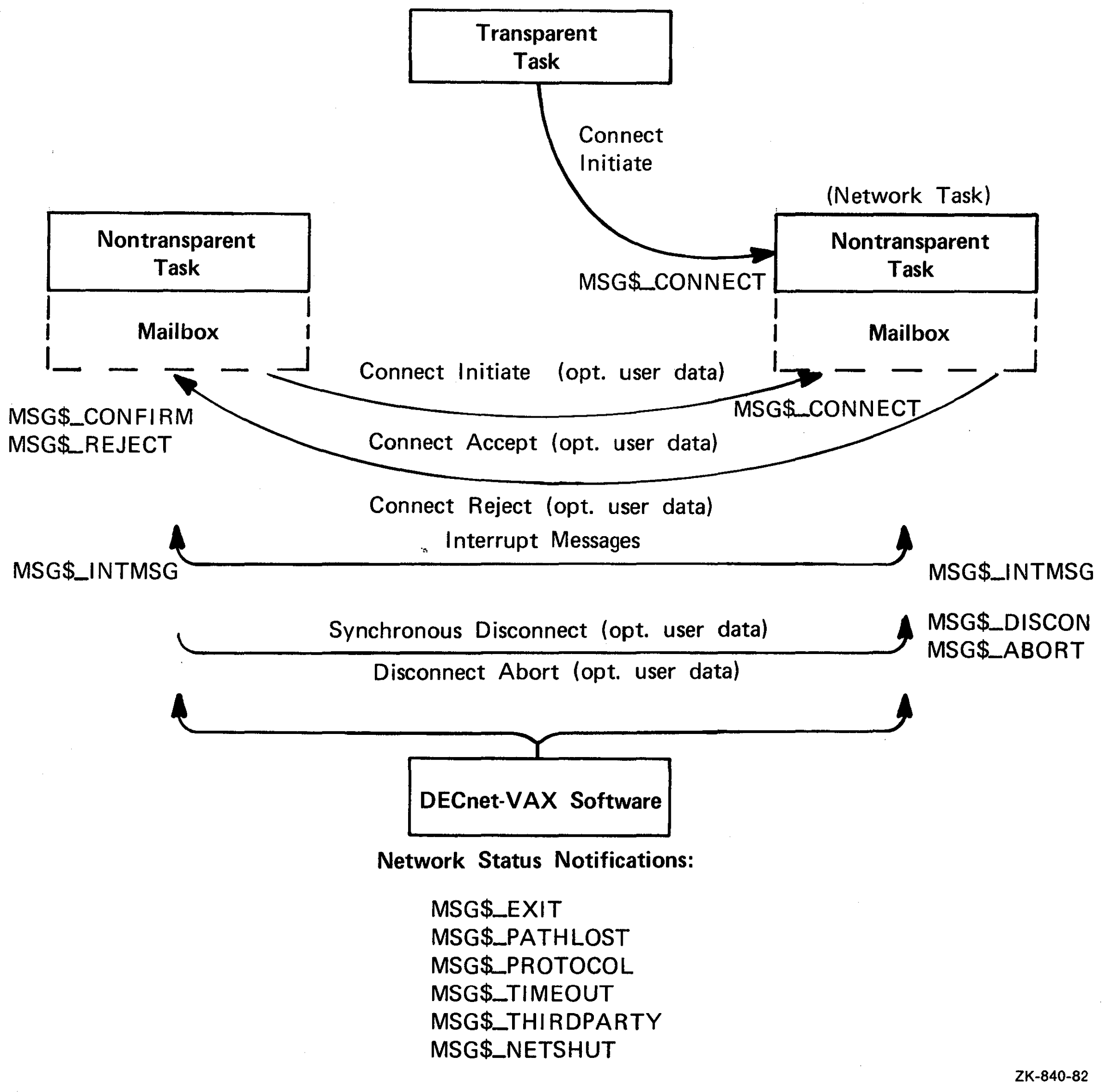


Figure 5-1: Mailbox Messages

A nontransparent task can also receive **network status notifications** via the mailbox. These notifications apply to physical and logical link conditions over the network. For example, DECnet-VAX software can notify a nontransparent task of the following conditions:

- Third-party disconnections (see the DECnet-VAX System Manager's Guide)
- Network software- and hardware-related problems
- Processes exiting before I/O completion
- Connection request timeouts

5.3 INITIATING A LOGICAL LINK CONNECTION

Regardless of whether you access the network transparently or nontransparently, you must establish a communication link to the remote node on which the target task runs before any message exchange can take place. You establish the link by issuing a source task call that requests a logical link connection. (To clarify, the term **source task** refers to the task that initiates a logical link connection request. The term **target task** refers to the task with which you want to communicate. The interaction that takes place prior to establishing a logical link is termed a **handshaking sequence.**)

5.3.1 The Handshaking Sequence

Upon receiving a call that requests a logical link connection, the local DECnet-VAX initiates a handshaking sequence with the target task. The following information is supplied in a connection request:

- An I/O channel: The I/O channel serves as the path over which messages are sent and received by the source program.
- The identification of the target node: Every node in a network has an identifier that distinguishes it from all other nodes in the network. Transparent communication uses a task specification string to indicate the name of the target node (see Chapter 2). Nontransparent communication requires a user-generated data structure called the **Network Connect Block** (NCB) which includes a task specification string (see Chapter 7).
- An object type descriptor (see Chapter 2).
- Access control information (optional; see Chapter 2).
- Optional user data: Nontransparent tasks have the option of sending up to 16 bytes of data to the target program. (See the information on NCBs in Chapter 7.)

Higher-level language tasks can use standard file opening statements to request a logical link connection to a remote task. The following examples in VAX-11 FORTRAN, VAX-11 BASIC, VAX-11 PL/I, VAX-11 PASCAL, and VAX-11 COBOL show how to specify a target task, TEST4, running on node TRNTO:

```
FORTRAN    OPEN (UNIT=7,NAME='TRNTO::"TASK=TEST4"',TYPE='NEW')

BASIC      OPEN 'TRNTO::"TASK=TEST4"' AS FILE #7

PL/I       OPEN FILE(OUTPUT) TITLE ('TRNTO::"TASK=TEST4"');

PASCAL     OPEN (PARTNER,'TRNTO::"TASK=TEST4"',NEW);

COBOL      SELECT PARTNER ASSIGN TO "TRNTO::""TEST=TASK4""".
           OPEN OUTPUT PARTNER.
```

The RMS service call equivalent to these higher-level language statements is the $OPEN call. System service calls used to request logical link connection are described in Chapters 6 and 7.

It is important to note that once you issue a call that uses either a task specification string or an NCB, you access the network and, by definition, DECnet-VAX software.

5.4 COMPLETING THE LOGICAL LINK CONNECTION

As part of the handshaking sequence that takes place between the source and target tasks, the target task completes the logical link connection in two steps. First, the DECnet software at the remote node processes the inbound logical link connection request and then the target task either accepts or rejects the link. These steps are performed differently, depending on whether the target task uses transparent or nontransparent I/O. In the following discussion, it is assumed that the remote node is VAX/VMS. If the remote node on which your target task runs is not a VAX/VMS system, you should refer to the DECnet documentation for that system.

5.4.1 Completing the Connection Transparently

If the target task is transparent, the software at the remote node checks the access control information supplied in the connection request call. DECnet-VAX software creates a process in which the LOGINOUT image runs. The name of this process is a concatenation of the name of the object connected to and the logical link number (for example, MAIL_65218). The LOGINOUT image verifies the access control information against the user name and password contained in the User Authorization File (UAF) at the remote node.

If the access control information is valid, LOGINOUT creates the logical name SYS$NET in the process logical name table for subsequent use by the target task. The equivalence string for SYS$NET is a special form of file specification string that contains information identifying the source task which initiated the logical link connection. The LOGIN.COM file associated with the access control string is then run. Finally, the command procedure file (taskname.COM) for starting the remote task is executed.

Prior to your accessing the remote node, the System Manager must have created the appropriate account in the UAF (refer to the information on access control in Chapter 2.) In addition, the command procedure file (taskname.COM) for starting the remote task must exist in the directory associated with the account identified by the access control information. For a description of the command procedure taskname.COM, see Section 5.4.3. Command procedures for objects existing in the OBJECT data base (which is created using NCP commands) are located in the SYS$SYSTEM directory.

To complete the logical link, the target task performs a file opening operation using the logical name SYS$NET to establish a communications path back to the source task. The following examples in VAX-11 FORTRAN, VAX-11 BASIC, VAX-11 PL/I, VAX-11 PASCAL, and VAX-11 COBOL show how to specify SYS$NET:

FORTRAN	`OPEN (UNIT=2,NAME='SYS$NET',TYPE='OLD'))`
BASIC	`OPEN "SYS$NET" AS FILE #2`
PL/I	`OPEN FILE(INPUT) TITLE ('SYS$NET');`
PASCAL	`OPEN (PARTNER,'SYS$NET',OLD);`
COBOL	`SELECT PARTNER ASSIGN TO "SYS$NET".` `OPEN INPUT PARTNER.`

The RMS service call equivalent to these higher-level language statements is the $OPEN call. System service calls for accepting the logical link are described in Chapters 6 and 7.

5.4.2 Completing the Connection Nontransparently

If the target task is nontransparent, then one of several things may occur. If the task has not declared itself a **network task** (and is therefore eligible to accept only one connection request at a time), then the DECnet software at the remote node performs the access checking procedure described in Section 5.4.1. After it starts, the target task retrieves the connection information by translating the logical name SYS$NET using the $TRNLOG system service call (see Chapter 7).

If the target task declares itself as an active network task, then DECnet-VAX software places all connection requests addressed to the task in the mailbox associated with the channel being used. The first message in the mailbox is the Network Connect Block (NCB) from the original connection request that started the task. This message appears in the mailbox after channel assignment and name declaration occur. Once the task declares a network name, subsequent inbound connection requests are not checked by the remote node to verify access control. (Note that if the task is started without being part of a DECnet operation, access control is never checked.) Chapter 7 describes in more detail the nontransparent process of completing the logical link connection.

After examining the incoming connection request, the target task either accepts or rejects the request, and optionally it can send 1 to 16 bytes of data back to the source task at the same time that it responds to the logical link connection request. Furthermore, a library routine, LIB$ASN_WTH_MBX, that assigns a channel and associates a unique mailbox, can be used when accepting the connection if no optional data is returned (see Section 7.2).

5.4.3 Command Procedures Used in Task-to-Task Communication

As described above, once the access control information is verified, both the LOGIN.COM command procedure for the accessed account and the taskname.COM command procedure in the default directory under that account are executed.

On a VAX/VMS system, jobs are classified as interactive, batch, or network. Inclusion of the following command in the LOGIN.COM file avoids the execution of DCL commands applicable to the interactive mode that are not required for task-to-task communication.

```
     $ IF F$MODE() .NES. "INTERACTIVE" THEN EXIT.
```

The command procedure file taskname.COM must contain at minimum a RUN command to cause the target task image to be executed. It may also contain terminal assignments for debugging purposes (for example, DBG$INPUT and DBG$OUTPUT). There are no restrictions on the type of commands that you can have in this file. A sample command procedure DEMO.COM to run the target task TASK20.EXE might contain the following commands:

```
     $ RUN USER:[DEMO.TEMP]TASK20.EXE
     $ PURGE/KEEP=2 SYS$LOGIN:DEMO.LOG
     $ LOGOUT/BRIEF
```

A sample command procedure TEST.COM used to debug the target task TEXT.EXE might contain the following commands:

```
     $ ASSIGN _TTA7: DBG$INPUT
     $ ASSIGN _TTA7: DBG$OUTPUT
     $ RUN/DEBUG SYS$LOGIN:TEST.EXE
     $ PURGE/KEEP=2 SYS$LOGIN:TEST.LOG
```

The debug terminal must not currently be assigned to any process. Otherwise, the command procedure will exit with an error that causes the logical link connection to the object to be rejected.

5.5 EXCHANGING MESSAGES

After DECnet-VAX creates a logical link, the two tasks are ready to exchange messages. The exchange of messages can take place only if the two tasks **cooperate** with each other. In other words, for each message sent by a task, the receiving task must issue a corresponding call to receive the message. Also, you must decide which task will disconnect the link. In addition, if the tasks are nontransparent, they must agree on the optional data to be passed. Because either task can now send and receive messages, the distinction between source and target must be redefined. In the context of an established logical link, the task sending a message is the source and the task receiving it is the target.

DECnet-VAX distinguishes between two types of messages: data messages and mailbox messages. For both transparent and nontransparent communication, data messages are the normal mode of information exchange. Nontransparent communication gives you the additional capability of receiving mailbox messages, such as interrupt messages, messages resulting from some DECnet operation (including optional user data), and network status notifications.

5.5.1 Data Messages

Whether you access the network transparently or nontransparently, DECnet-VAX sends data messages over a logical link in response to a set of send and receive calls issued by the source and target tasks. For higher-level language tasks, use standard read and write calls to send and receive data messages. In Example 5-1, the two FORTRAN tasks use read and write calls to exchange information. The equivalent RMS service calls are $GET and $PUT. System service calls to send and receive data messages are described in Chapters 6 and 7.

5.5.2 Mailbox Messages

Nontransparent communication frequently involves using a mailbox to obtain network-specific information. A task may receive the following types of messages in its mailbox:

- Messages that DECnet generates when a task initiates any of the network operations listed below (a VAX/VMS task issues system service calls to initiate these operations; these calls also permit it to place optional user data in the mailbox of the other task):

  - When one task requests a logical link connection, a notification message (and optional user data) may be placed in the mailbox of the target task.
  - When a target task accepts or rejects the logical link connection request, a notification message (and optional user data) is placed in the mailbox of the source task.
  - When one task synchronously disconnects or aborts a logical link, a notification message (and optional user data) is placed in the mailbox of the task from which it is disconnecting.

- Network status notification messages that inform a task of some unusual network occurrence (such as a third-party disconnect).
- Interrupt messages sent by the other task.

5.6 TERMINATING THE COMMUNICATION PROCESS

The termination of a logical link signals the end of the communication process between two tasks. In transparent task-to-task communication using higher-level language statements or RMS service calls, either task can issue a file closing statement to break the link. To terminate the link properly, it is recommended that the receiver, and not the transmitter, of the final message issue the close call to disconnect the link. The link termination process is complete when the other task issues a file closing call. In transparent communication using system service calls, the $DASSGN system service call causes the link to be terminated (see Chapter 6).

If you are using system service calls in nontransparent task-to-task communication, you can terminate I/O operations over a channel in one of three ways (as described in detail in Chapter 7):

- **Synchronous Disconnect ($QIO)** - This form of disconnection indicates to the remote receiver that all messages sent by the local transmitter have been acknowledged at the Network Services Protocol (NSP) level. This does not guarantee that the user of NSP received the data.
- **Disconnect Abort ($QIO)** - This form of disconnection indicates to the remote receiver that all messages sent have not necessarily been received. To ensure that all messages are received before the link is disconnected, the sender must cancel I/O on the channel before issuing the abort call.
- **Deassign Channel and Terminate Link ($DASSGN)** - This form of disconnection deassigns the channel and immediately disconnects the link.

Note that after either a synchronous disconnect or a disconnect abort, you can issue a new connection request since you did not deassign the I/O channel but merely deaccessed the link.

When a nontransparent task terminates the communication process, a notification message indicating that the link is disconnected is placed in mailbox of the task affected. In addition, a nontransparent task can send up to 16 bytes of optional user data which is also placed in the mailbox.

Note that by their nature, disconnect operations are of little value in guaranteeing to both partners that communication is complete. Therefore, it is recommended that the communicating tasks agree on a protocol for terminating communication. In general, the receiver and not the transmitter of the final message should disconnect the logical link.

CHAPTER 6

TRANSPARENT TASK-TO-TASK COMMUNICATION USING SYSTEM SERVICES

This chapter describes the system service calls and functions that you can use to perform transparent task-to-task communication over the network. In general, transparent communication allows you to:

- Create a logical link between tasks
- Send and receive data messages
- Terminate the logical link at the end of the message dialog

This chapter defines the system service calls for these functions and develops an example to illustrate their use.

The general concepts implicit in DECnet-VAX task-to-task communication are covered in Chapter 5. You should also be familiar with the QIO-related material in the VAX/VMS System Services Reference Manual. The use of higher-level language statements and RMS service calls in transparent task-to-task communication is described in Chapter 5.

6.1 SYSTEM SERVICE CALLS FOR TRANSPARENT COMMUNICATION

Transparent task-to-task communication uses a set of system service calls available with the VAX/VMS operating system. Table 6-1 summarizes these calls and their specific network-related functions. The $QIO calls are distinguished by function code. Section 6.7 presents the format of these calls in more detail.

Table 6-1: Transparent Task-to-Task Communication System Service Summary

Call	Function
$ASSIGN	Request a logical link connection
$DASSGN	Terminate a logical link
$QIO (IO$_READVBLK)	Receive a message
$QIO (IO$_WRITEVBLK)	Send a message

The system service calls summarized in Table 6-1 allow you to perform task-to-task communication in much the same way as you would perform normal I/O operations. Use the $ASSIGN call to assign a logical link I/O channel to a "device," which in this case is a task that behaves like a full-duplex record-oriented device. You can perform read and write operations with this task in either a synchronous or asynchronous manner. To exchange messages, use the Queue I/O (QIO) Requests supported by DECnet-VAX. When all communication completes, use the $DASSGN system service call to deassign the channel and thereby disconnect the logical link.

6.2 REQUESTING A LOGICAL LINK

To request a logical link and assign an I/O channel, use the $ASSIGN system service. When you issue this call, you must include a task specifier for the remote node on which the cooperating task runs. The task specifier identifies the remote node and the target task to which you want to establish a logical link. (Refer to Chapter 2 for the format of the task specification string.)

For example, for the network model described in Chapter 1, you could establish a logical link to target task TEST2 on node TRNTO to perform task-to-task communication. To create this link, code the following VAX-11 MACRO statements in your source program:

```
TARGET:         .ASCID  /TRNTO::"TASK=TEST2"/
NETCHAN:        .BLKW   1         ; channel number returned here
    .
    .
    .
$ASSIGN_S       DEVNAM=TARGET,CHAN=NETCHAN
```

For debugging or for symmetry, you can develop and run the target task on the local node. Use the local node name plus two colons (::) to connect to the local node. Alternatively, you can use node number 0 plus double colons (::) to connect to the local node.

Once you establish a logical link, you refer to the assigned channel in any succeeding call in the MACRO program, either to send or receive messages, or to deassign the channel and terminate the logical link.

Until the connection operation completes, the process is in Local Event Flag Wait (LEF) state in KERNEL mode. Therefore, pressing CTRL/Y will not return the process to DCL status. The maximum amount of time that the process will wait in this state is specified by the NCP command SET EXECUTOR with the INCOMING TIMER parameter. If this timer cannot be set to an acceptable value, tasks that accept commands from the terminal should use $QIO (IO$_ACCESS) instead of the transparent $ASSIGN call to initiate logical links (see Section 7.8.2).

6.3 COMPLETING THE LOGICAL LINK CONNECTION

The target task completes the logical link by specifying the logical name SYS$NET as the devnam argument for the $ASSIGN system service. For example:

```
LOGNAM:         .ASCID  /SYS$NET/
NETCHAN:        .BLKW   1         ; channel number returned here
    .
    .
    .
$ASSIGN_S       DEVNAM=LOGNAM,CHAN=NETCHAN
```

Issue these calls in the target task to complete the logical link connection. The target task also specifies a channel to be used in subsequent system service calls.

In this chapter, it is assumed that the remote node is a VAX/VMS system. If the remote node on which the target task runs is other than VAX/VMS, you should refer to the related DECnet documentation.

6.4 EXCHANGING MESSAGES

After DECnet-VAX software establishes a logical link with the target task, either task can then send or receive messages. However, they must cooperate with each other: for each message sent with the $QIO (IO$_WRITEVBLK), the other task must issue a corresponding $QIO (IO$_READVBLK) to receive the message.

You must ensure that the tasks allocate enough buffer space for receiving the messages; if they do not, a DATAOVERUN error will occur. You must also ensure that the end of the dialog can be determined. Finally, one of the two tasks must disconnect the logical link. To terminate a logical link properly, the receiver, and not the transmitter, of the final message should break the link.

6.5 TERMINATING THE LOGICAL LINK

For transparent task-to-task communication, use the $DASSGN system service call to deassign the channel and break off the logical link with the cooperating task. This call terminates all pending calls for sending and receiving messages, aborts the link immediately, and frees the channel associated with that logical link.

6.6 STATUS AND ERROR REPORTING

When a system service completes execution, a status value is always returned. The $ASSIGN, $DASSGN, and $QIO system services place the return status information in register 0 (R0). For the $QIO system service, a successful status return indicates only that the request was successfully queued. All I/O completion status information is placed in the I/O status block (IOSB). For example, a $QIO system service read request to a task might be successful (status return is SS$_NORMAL) yet fail because the link was disconnected (I/O status return is SS$_LINKABORT). The return status codes shown in the following sections may be returned both in R0 and in the IOSB.

The VAX/VMS System Services Reference Manual and the VAX/VMS I/O User's Guide both describe the use of asynchronous system traps (ASTs), IOSBs, and event flags.

6.7 SYSTEM SERVICE CALL SUMMARY

The following subsections describe the VAX/VMS system services you can use for transparent intertask communication. Each subsection describes the use of the call, its format, the arguments associated with the call, and the return status information. Appendix C lists the entire set of network system service error messages.

6.7.1 $ASSIGN (I/O Channel Assignment)

The $ASSIGN system service assigns a channel to refer to the logical link. You can then use the channel returned in the chan argument in any succeeding call to send or receive a message, or to deassign the channel and thereby terminate the logical link.

Format

```
    $ASSIGN devnam ,chan ,[acmode]
```

Arguments

devnam	address of a quadword descriptor of a character string that identifies the remote task. The string will contain either: • A task specification string if the call is by the source task. (Refer to Chapter 2 for the task specification string format.) Both the string and its descriptor must be in read/write storage. • SYS$NET if the call is by the target task. (SYS$NET is a logical name.)
chan	address of a word that will receive the assigned channel number. You use this channel number to send a message to a remote task, receive a message from a remote task, or to abort the logical link.
acmode	access mode to be associated with this channel. The most privileged access mode used is the access mode of the caller. You can perform I/O operations on the channel only from equal or more privileged access modes.

Return Status

SS$_REMOTE	The service completed successfully. (A logical link was established with the target task.)
SS$_CONNECFAIL	The connection to a network object timed out or failed.
SS$_DEVOFFLINE	The physical link is shutting down.
SS$_FILALRACC	A logical link already exists on the channel.
SS$_INSFMEM	There is not enough system dynamic memory to complete the request.
SS$_INVLOGIN	The access control information was found to be invalid at the remote node.
SS$_IVDEVNAM	The task specifier has an invalid format or content.
SS$_LINKEXIT	The network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOLINKS	No logical links are available. The maximum number of logical links as set for the NCP executor MAXIMUM LINKS parameter was exceeded.

SS$_NOPRIV	The issuing task does not have the required privilege to perform network operations or to confirm the specified logical link.
SS$_NOSUCHNODE	The specified node is unknown.
SS$_NOSUCHOBJ	The network object number is unknown at the remote node; or for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	The remote node could not recognize the login information supplied with the connection request.
SS$_PROTOCOL	A network protocol error occurred, most likely because of a network software error.
SS$_REJECT	The network object rejected the connection.
SS$_REMRSRC	The link could not be established because system resources at the remote node were insufficient.
SS$_SHUT	The local or remote node is no longer accepting connections.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).
SS$_TOOMUCHDATA	The task specified too much optional or interrupt data.
SS$_UNREACHABLE	The remote node is currently unreachable.

6.7.2 $QIO (Sending a Message to a Target Task)

The $QIO system service with a function code of IO$_WRITEVBLK sends a message to a target task. The $QIO call initiates an output operation by queuing a request to the channel associated with the logical link. (Alternatively, you could use the $QIOW system service to perform the same operation.)

Format

```
{ $QIO  }[efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,p1 ,p2
{ $QIOW }
```

Arguments

efn	number of the event flag to be set at request completion.
chan	a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_WRITEVBLK
iosb	address of a quadword I/O status block that is to receive the completion status.

astadr	entry point address of an AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	AST parameter to be passed to the AST completion routine.
p1	buffer address.
p2	buffer length in bytes.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_ABORT	The I/O request has been aborted by a $DASSGN or $CANCEL.
SS$_CANCEL	The I/O on this channel has been cancelled.
SS$_FILNOTACC	No logical link is associated with the channel.
SS$_INSFMEM	Enough memory to buffer the message could not be allocated.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This is most likely due to a network software error.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).

6.7.3 $QIO (Receiving a Message from a Target Task)

The $QIO system service with a function code of IO$_READVBLK receives a message from a target task. The $QIO call initiates an input operation by queuing a request to the channel associated with the logical link. (Alternatively, you could use the $QIOW system service to perform the same operation.)

Format

```
{ $QIO  }[efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,p1 ,p2
{ $QIOW }
```

Arguments

efn	number of the event flag to be set at request completion.
chan	a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.

func	IO$_READVBLK
iosb	address of a quadword I/O status block that is to receive the completion status.
astadr	entry point address of an AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	AST parameter to be passed to the AST completion routine.
p1	buffer address.
p2	buffer length in bytes.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_ABORT	The I/O request has been aborted by a $DASSGN or $CANCEL.
SS$_CANCEL	The I/O on this channel has been cancelled.
SS$_DATAOVERUN	More bytes were sent than could be received in the supplied buffer.
SS$_FILNOTACC	No logical link is associated with the channel.
SS$_INSFMEM	Enough memory to buffer the message could not be allocated.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This is most likely due to a network software error.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).

6.7.4 $DASSGN (Terminating a Logical Link)

The $DASSGN system service terminates all pending operations to send and receive data, disconnects the logical link immediately, and frees the channel associated with that link. Either task can terminate the logical link by calling $DASSGN.

Format

```
$DASSGN chan
```

Arguments

chan	a word containing the channel number to the logical link you want disconnected. Use the same channel number returned previously in the $ASSIGN call.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_IVCHAN	An invalid channel number was specified.
SS$_NOPRIV	The specified channel is not assigned; or it was assigned from a more privileged access mode.

6.8 PROGRAMMING EXAMPLE OF TRANSPARENT COMMUNICATION

Example 6-1 illustrates the use of these system service calls for transparent task-to-task communication. TRANA is a MACRO source task on the local node that communicates with a target task, TRANB, on node TRNTO. The source task sends a connection request to the remote node whereupon the target task is started by the command file TRANB.COM. Once the logical link connection is made, the source task sends a message to the target task, which in turn responds with a message and then waits for additional message traffic. The source task drives the communication process. Once the source task receives a response from the target task, it disconnects the link and exits, which causes the target task to exit also, thereby terminating the communication process.

Example 6-1: Transparent Task-to-Task Communication Using System Services

TRANA

```
        .TITLE  TRANA   - SOURCE TASK USING TRANSPARENT I/O
        .IDENT  /V1.0/
        .SBTTL  WRITABLE_DATA
        .PSECT  TRANA$DATA      SHR,NOEXE,RD,WRT,BYTE

NETCHAN:.BLKW   1                               ; Network channel
IOSBUF: .BLKQ   1                               ; I/O status block
TARGET: .ASCID  /TRNTO"MALIK KARL"::"TASK=TRANB"/  ; Task spec (& descriptor)

SENDMSG:.ASCII  /SEND THIS STRING TO TRANB/     ; Output buffer
SENDMSG_SIZ=.-SENDMSG                           ; Output buffer size
RECVMSG:.BLKB   512                             ; Input buffer
RECVMSG_SIZ=.-RECVMSG                           ; Input buffer size

        .SBTTL  MAIN
        .PSECT  TRANA$CODE      NOSHR,EXE,RD,NOWRT,BYTE
        .ENTRY  START,^M<>              ;Entry point from exec
```

```
;
; Request a logical link to the target task and assign an I/O channel.
;
        $ASSIGN_S -                     ; Assign a channel to target task
                DEVNAM=W^TARGET,-       ; Address of device name descriptor
                CHAN=W^NETCHAN          ; Adr to receive channel #
        BLBC    R0,EXIT                 ; Branch on failure
;
; Send a message to the target task.
;
        $QIOW_S -                       ; Issue transmit request
                EFN=#1,-                ; Use local event flag #1
                CHAN=W^NETCHAN,-        ; Use assigned channel
                FUNC=S^#IO$_WRITEVBLK,- ; Write virtual block
                IOSB=W^IOSBUF,-         ; Address of I/O status block
                P1=W^SENDMSG,-          ; Address of output buffer
                P2=S^#SENDMSG_SIZ       ; Size of output buffer
        BLBC    R0,EXIT                 ; Branch on failure
        MOVZWL  W^IOSBUF,R0             ; Get completion status
        BLBC    R0,EXIT                 ; Branch on failure
;
; Receive a message from the target task.
;
        $QIOW_S -                       ; Issue receive request
                EFN=#1,-                ; Use local event flag #1
                CHAN=W^NETCHAN,-        ; Use assigned channel
                FUNC=S^#IO$_READVBLK,-  ; Read virtual block
                IOSB=W^IOSBUF,-         ; Address of I/O status block
                P1=W^RECVMSG,-          ; Address of input buffer
                P2=#RECVMSG_SIZ         ; Size of input buffer
        BLBC    R0,EXIT                 ; Branch on failure
        MOVZWL  W^IOSBUF,R0             ; Get completion status
        BLBC    R0,EXIT                 ; Branch on failure
;
; Abort the logical link.
;
        $DASSGN_S -                     ; Deassign the channel
                CHAN=W^NETCHAN          ; Adr of word containing channel #
;
; Exit with status (in R0).
;
EXIT:   $EXIT_S R0                      ; Exit with status to be displayed
                                        ;  on error condition

         .END    START                   ; Image transfer address
```

TRANB

```
         .TITLE  TRANB - TARGET TASK USING TRANSPARENT I/O
         .IDENT  /V1.0/
         .SBTTL  WRITABLE_DATA
         .PSECT  TRANB$DATA      SHR,NOEXE,RD,WRT,BYTE

 NETCHAN:.BLKW   1                       ; Network channel
 IOSBUF: .BLKQ   1                       ; I/O status block
 RECVMSG:.BLKB   512                     ; Input buffer
 RECVMSG_SIZ=.-RECVMSG                   ; Input buffer size
 LOGNAM: .ASCID  /SYS$NET/               ; Logical name & descriptor
 SENDMSG:.ASCII  /REPLY TO TRANA/        ; Output buffer
 SENDMSG_SIZ=.-SENDMSG                   ; Output buffer size

         .SBTTL  MAINLINE
         .PSECT  TRANB$CODE      NOSHR,EXE,RD,NOWRT,BYTE
         .ENTRY  START,^M<>              ;Entry point from exec
```

```
;
; Complete the logical link connection (that TRANA requested).
;
        $ASSIGN_S -                     ; Assign channel to SYS$NET
                DEVNAM=W^LOGNAM,-       ; Descriptor of SYS$NET
                CHAN=W^NETCHAN          ; Store channel #
        BLBC    R0,EXIT                 ; Branch on failure
LOOP:
;
; Receive message from source task.
;
        $QIOW_S -                       ; Issue receive request
                EFN=#1,-                ; Use local event flag #1
                CHAN=W^NETCHAN,-        ; Use assigned channel
                FUNC=S^#IO$_READVBLK,-  ; Read virtual block
                IOSB=W^IOSBUF,-         ; Address of I/O status block
                P1=W^RECVMSG,-          ; Address of input buffer
                P2=#RECVMSG_SIZ         ; Size of input buffer
        BLBC    R0,EXIT                 ; Branch on failure
        MOVZWL  W^IOSBUF,R0             ; Get completion status
        CMPW    S^#SS$_ABORT,R0         ; Was logical link aborted?
        BEQL    DONE                    ; Branch if yes
        BLBC    R0,EXIT                 ; Branch on failure
;
; Send message to source task.
;
        $QIOW_S -                       ; Issue transmit request
                EFN=#1,-                ; Use local event flag #1
                CHAN=W^NETCHAN,-        ; Use assigned channel
                FUNC=S^#IO$_WRITEVBLK,- ; Write virtual block
                IOSB=W^IOSBUF,-         ; Address of I/O status block
                P1=W^SENDMSG,-          ; Address of output buffer
                P2=S^#SENDMSG_SIZ       ; Size of output buffer
        BLBC    R0,EXIT                 ; Branch on failure
        MOVZWL  W^IOSBUF,R0             ; Get completion status
        BLBC    R0,EXIT                 ; Branch on failure
        BRB     LOOP                    ; Reissue the read
;
; Logical link was aborted.
;
DONE:   $DASSGN_S -                     ; Deassign the channel
                CHAN=W^NETCHAN          ; Address of channel #
;
; Exit with status (in R0).
;
EXIT:   $EXIT_S R0                      ; Exit with status to be displayed
                                        ;  on error condition

        .END    START                   ; Image transfer address
```

CHAPTER 7

NONTRANSPARENT TASK-TO-TASK COMMUNICATION USING SYSTEM SERVICES

This chapter describes the system service calls and functions that you use to perform nontransparent task-to-task communication. In general, the underlying principles of nontransparent task-to-task communication are similar to those of transparent communication. However, several extensions to the system services described in Chapter 6 allow you to use network-specific features for network operations. These extensions allow you to do the following:

- Create and use mailboxes for receiving messages, including network status notifications
- Declare a task as a network task, thus enabling it to process multiple inbound logical link connection requests
- Send connection requests, optionally with user data
- Accept or reject a connection request, optionally with user data
- Communicate between a transparent and a nontransparent task
- Send or receive an interrupt message
- Synchronously disconnect or disconnect abort a logical link, optionally with user data

This chapter defines the system service calls for these functions and presents an example to illustrate their use.

The general concepts implicit in DECnet-VAX task-to-task communication are covered in Chapter 5. You should also be familiar with the material contained in the VAX/VMS System Services Reference Manual and the VAX/VMS I/O User's Guide.

7.1 SYSTEM SERVICE CALLS FOR NONTRANSPARENT COMMUNICATION

Nontransparent task-to-task communication over the network uses a set of system service calls available under the VAX/VMS operating system. Table 7-1 summarizes these calls and their network-related functions. The $QIO calls are distinguished by function code. (Section 7.8 presents the format of these calls in more detail.)

Table 7-1: Summary of System Service Calls for Nontransparent Task-to-Task Communication

Call	Function
$ASSIGN	Assign an I/O channel
$CANCEL	Cancel I/O on a channel
$CREMBX	Create a mailbox
$DASSGN	Abort the logical link connection (deassigning an I/O channel)
$GETDVI	Get information on device or volume
$QIO (IO$_ACCESS)	Request a logical link connection
$QIO (IO$_ACCESS)	Accept a logical link connection request
$QIO (IO$_ACCESS!IO$M_ABORT)	Reject a logical link connection request
$QIO (IO$_ACPCONTROL)	Assign a network name to a task eligible to accept multiple inbound connection requests
$QIO (IO$_DEACCESS!IO$M_ABORT)	Abort the logical link connection
$QIO (IO$_DEACCESS!IO$M_SYNCH)	Synchronously disconnect a logical link
$QIO (IO$_READVBLK)	Receive a message
$QIO (IO$_WRITEVBLK)	Send a message
$QIO (IO$_WRITEVBLK!IO$M_INTERRUPT)	Send an interrupt message
$TRNLOG	Translate logical names

7.2 ASSIGNING A CHANNEL TO _NET: AND CREATING A MAILBOX

To prepare for nontransparent task-to-task communication, you need to assign a channel just as you would for transparent communication. In addition, to take advantage of optional network protocol features, you can create a mailbox.

For task-to-task communication, you must assign a channel to a pseudo-device called _NET:. Use the $ASSIGN system service call for this purpose. This call normally contains a reference to a mailbox, thereby associating it with the channel assigned to _NET:. If you use a mailbox, you must create the mailbox before assigning a channel to _NET:. It is important to note that this use of the $ASSIGN system service differs from its use for transparent communication. Assigning a channel to _NET: does not transmit a logical link connection request to the remote node. Instead, the channel to _NET: provides a communication path to DECnet software. A separate $QIO call (IO$_ACCESS function using the same channel) must be used to request a logical link to the remote task.

To take advantage of optional network protocol features, you can create a mailbox to receive messages on behalf of logical link operations. For example, the mailbox receives a message indicating whether the cooperating task accepted or rejected a connection request issued by the source task. Use the $CREMBX system service to create a mailbox for these purposes. In the event that your application does not use mailbox messages, you need not create a mailbox.

For convenience, the run-time library routine LIB$ASN_WTH_MBX can be used to create a temporary mailbox, assign a channel to it, and assign a channel to _NET:. This routine creates a unique mailbox such that multiple copies of the task using it will in effect use different mailboxes. If you were to create a mailbox with a logical name, all tasks would use the same mailbox and thereby interfere with each other's mailbox messages. The program example in Section 7.9 illustrates a use of this routine. Refer to the VAX-11 Run-Time Library Reference Manual for a complete description of this routine.

7.2.1 Mailbox Message Format

The mailbox receives information specific to nontransparent communication with a remote task. Figure 7-1 illustrates the general format of the mailbox message.

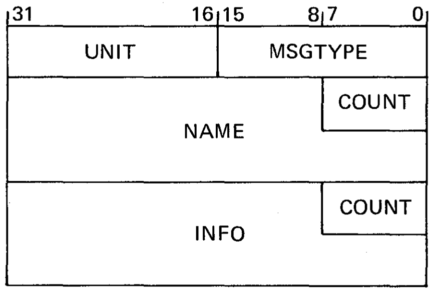


Figure 7-1: Mailbox Message Format

Notes to Figure 7-1:

MSGTYPE	Contains a code that identifies the message type.
UNIT	Contains the binary unit number of the device for which the message applies.
COUNT NAME	Contain a counted ASCII string that gives the name of the device for which the message applies. The $ASSIGN system service creates devices having names of "NET".
COUNT INFO	Contain a counted ASCII string of information which depends on the message type.

Appendix D summarizes the mailbox messages that pertain to nontransparent task-to-task communication.

7.3 REQUESTING A LOGICAL LINK CONNECTION

Once you assign the I/O channel, you can then request a logical link connection. To request a logical link connection to the target task, use the $QIO system service with a function code of IO$_ACCESS. You must identify the target task in the $QIO call. Use a Network Connect Block (NCB) to specify the target task identification string. In addition, you can optionally send 1 to 16 bytes of data in the NCB. The NCB must be in read/write storage. The format of the NCB is discussed in Section 7.3.1.

Once the source task issues the connection request, it can then issue a $QIO call with a function code of IO$_READVBLK to read its mailbox. Checking the contents of the mailbox is one way to determine whether the target task accepted or rejected the connection request. The mailbox will contain either MSG$_CONFIRM or MSG$_REJECT, possibly with optional data in the mailbox buffer.

If specified, the IOSB argument of the $QIO (IO$_ACCESS) call will also contain the result of the connection request operation. Section 7.8.2 provides a complete list of I/O status messages for this call.

Note that you must read the mailbox to inspect any optional data sent from the target task upon accepting or rejecting the connection request.

7.3.1 Network Connect Block

The Network Connect Block (NCB) is a user-generated data structure that contains the information necessary to request a logical link connection, or to accept or reject a logical link connection request.

Figure 7-2 is an example of an NCB you could use when issuing a connection request call.

The significance of the information contained in the NCB block varies, depending on the type of call in which it is used. If the call is an outbound connection request with no optional data, items 2, 3, 4, and 5 of the block are not required. If the call is a connect accept operation and no optional data is sent, then items 4 and 5 are not required. Item 5 is meaningful only to the receiver of a connection request.

The example in Figure 7-2 illustrates an NCB that you could use when issuing a connection request call.

```
1.  With optional data (outbound connect):
                                    ❶      ❷
     NCB:      .ASCII  ?TRNTO::"TASK=TEST2/?
               .WORD   0❸
     OPTDATA:
               .ASCIC  /USERINFO/
               .BLKB 17-<.-OPTDATA> ❹
               .ASCII  /"/
                   ❺
2.  Without optional data (outbound connect):
                                    ❶
     NCB:        .ASCII   ?TRNTO::"TASK=TEST2"?
```

Item	Function
❶	A valid task specification string. (Refer to Chapter 2 for the task specification format.) For an inbound call with an NCB, the task name portion of the task specification string is a process ID if the remote node is a VAX/VMS system; if not, then the task name portion is a system-specific string that identifies an executable unit (for example, job or task). The task specification string must be a quoted string. Note that the final quotation mark of the task specification string follows the last item within the NCB.
❷	The slash character (/).
❸	One word. This word must be 0 for a connection request operation. For a connect accept or reject operation, this word contains an internal DECnet link identifier.
❹	Up to 16 bytes of optional data sent as a counted string. This string is stored in a fixed length field that is 17 bytes long. DECnet-VAX software ignores unused bytes.
❺	A destination descriptor. (This indicates how the connection was issued and is meaningful only to the task or object to which the connection is made. This information is useful where one program serves many functions and needs to know how it was invoked.) The maximum length for the destination descriptor is 19 bytes. The format is as follows: a. If byte 0 contains 0, then byte 1 is the binary value of the object number. b. If byte 0 contains 1, then byte 1 is the binary object number, and bytes 2 through 18 contain a counted task name. c. If byte 0 contains 2, then byte 1 is the binary object number, bytes 2 through 5 contain a UIC, the first two bytes of which contain a binary group code and the second two bytes contain a binary user code, and bytes 6 through 18 contain a counted task name.

Figure 7-2: Network Control Block Format

7.4 COMPLETING THE LOGICAL LINK

A nontransparent target task completes the logical link connection in one of several ways, depending upon whether the task can process multiple inbound connection requests or just a single request.

Furthermore, a nontransparent target task has the option of either accepting or explicitly rejecting a logical link request.

In the discussion that follows, it is assumed that the remote node is VAX/VMS. If the remote node on which your target task runs is other than VAX/VMS, you should refer to the related DECnet documentation.

7.4.1 Receiving a Connection Request

When a remote node receives a call requesting a logical link, the DECnet-VAX software constructs an NCB from the information contained in the call. At this point, one of two things occurs. If a process already running on the remote node has declared a network name or object number the same as the one identified in the NCB, then the software puts the NCB into its mailbox. If not, DECnet-VAX software creates a process that runs the LOGINOUT image. The LOGINOUT image verifies the access control information and, if it checks out, LOGINOUT equates SYS$NET to the NCB, invokes LOGIN.COM (if it exists), and starts the taskname.COM command file. The name of this command file is determined as follows:

- If the connection request identifies a numbered (nonzero) object, then the appropriate record is located in the Configuration Data Base and the name of the file is found in this record. (The file is assumed to be found in SYS$SYSTEM.)

- If the connection request identifies a named object with type 0 (TASK), then the file name is assumed to be the name of the task connected to (with a file type of .COM) and is assumed to be found in the default directory associated with the access control information.

Once executing, the target task can then determine whether to accept or explicitly reject the connection request. You can program the target task to base this assessment on the information contained in the NCB.

7.4.1.1 Receiving Single Connection Requests - A nontransparent target task can accept only one connection request at a time, unless it declares itself as a network task. The target task may retrieve the connection information by translating the logical name SYS$NET using the $TRNLOG system service call. Once the task retrieves the logical name, it may then decide whether to accept or explicitly reject the connection request.

Note that you need not translate SYS$NET if the following information is not required:

- The optional data in the Network Connect Block

- The identity of the connector

- The descriptor by which the connection was made

7.4.1.2 Receiving Multiple Connection Requests (Network Tasks) - A target task can accept multiple inbound connection requests only if it declares itself a known network task. To make this declaration, you must first use the $ASSIGN call to assign an I/O channel to _NET:. Then, use the $QIO system service with a function code of IO$_ACPCONTROL to assign a network name or object number to the task, making it eligible to process multiple inbound connection requests. This system service requires SYSNAM privilege. You must associate a mailbox with the channel if a name or number is to be declared.

Programs that have declared names or object numbers should be programmed to terminate when their mailboxes receive a MSG$_NETSHUT message. Such programs must be restarted when the network comes back up.

Once you declare the target task as an active network task, DECnet places all connection requests addressed to the task in the mailbox associated with the channel over which the ACP control function was issued. The target task retrieves connection requests from the mailbox by issuing the $QIO system service call with a function code of IO$_READVBLK. Note that the first message in the mailbox is the NCB from the original connection request that put the task into a state of execution. Once the task declares a network name or object number, subsequent inbound connection requests are not checked for their access control information. (However, if the network task is started separately from a DECnet operation, access control is never checked.)

Note that you can start programs that declare names or object numbers apart from the first inbound connection (that is, by a RUN command).

7.4.2 Accepting or Rejecting a Connection Request

As mentioned previously, the target task can either accept or explicitly reject a connection request. To accept a connection request, thus completing the logical link connection, use the $QIO system service with a function code of IO$_ACCESS. To reject the connection request, use the $QIO system service with the function code IO$_ACCESS!IO$M_ABORT. When it either accepts or rejects the connection request, the target task can also send 1 to 16 bytes of optional data within a modified NCB back to the source task.

7.5 EXCHANGING MESSAGES

Exchange of data messages between the two cooperating tasks is performed in the same way for both nontransparent and transparent communication. (Refer to Section 6.4 for information on using the system service calls $QIO (IO$_WRITEVBLK) and $QIO (IO$_READVBLK) to send and receive messages.) In addition, the following information on interrupt messages is particular to nontransparent communication.

7.5.1 Interrupt Messages

Either task can send a 1- to 16-byte interrupt message. You can use this method to send a message to a target task outside the normal flow of data messages. DECnet-VAX places the received interrupt message in the target task's mailbox. Use the $QIO system service with a function code of IO$_WRITEVBLK!IO$M_INTERRUPT to send the interrupt message. In order for the target task to be notified that an interrupt message has been placed in its mailbox, the task should issue a $QIO system service call with a function code of IO$_READVBLK to the mailbox. Specifying an AST routine in the $QIO system service call will cause the routine to be executed on receiving the interrupt message. (AST routines are described in the VAX/VMS System Services Reference Manual.)

7.6 DISCONNECTING OR ABORTING THE LOGICAL LINK

A nontransparent task can terminate communication with a remote task in one of two ways: either by disconnecting the link (synchronous disconnect or disconnect abort) or by deassigning the channel. In the first instance, you can issue a new connection request on the same channel since you do not deassign it. Regardless of the method you choose, you can send an optional message of 1 to 16 bytes of data with the $QIO call.

7.6.1 Synchronously Disconnecting a Logical Link

To synchronously disconnect a logical link, issue the $QIO system service with a function code of IO$_DEACCESS!IO$M_SYNCH. The channel is then free for subsequent communication with either the same or a different remote task.

A synchronous disconnection is useful for master/slave communication in which one task always sends data and its partner task always receives that data. If the receiver sees a synchronous disconnection notification, it knows that it received all the data that was sent. (The sender on the other hand is not guaranteed that its partner received the data.) Because this is the only guarantee provided by this operation, using this operation is discouraged in favor of a user-defined protocol to ensure completion of communication. In general, the receiver of the final message should break the logical link.

7.6.2 Aborting the Logical Link ($QIO Function)

To abort the logical link, issue the $QIO system service with a function code of IO$_DEACCESS!IO$M_ABORT. This form of disconnection indicates to the receiver that not all messages sent have necessarily been received. To ensure that all transmitted messages have been received, the task itself must terminate I/O operations on the link before instituting the DEACCESS function because this function never completes before all pending I/O operations complete. To do so, first issue the $CANCEL system service to terminate I/O operations over the link; then, issue the $QIO system service to terminate the link.

Note that after either a synchronous disconnect or a disconnect abort, you can issue a new connection request if you did not deassign the I/O channel.

If you issue the $CANCEL system service to a channel over which a network name or object has been declared, the declaration will be removed from the network data base.

7.6.3 Terminating the Logical Link ($DASSGN Function)

You can issue the $DASSGN system service call to deassign the channel and terminate the logical link immediately. You issue the $DASSGN call only after all communication between the tasks is complete. The call releases the I/O channel, disassociates the mailbox from the channel, and terminates the logical link immediately. This operation is equivalent to using $CANCEL followed by $QIO IO$_DEACCESS!IO$M_ABORT.

7.7 STATUS AND ERROR REPORTING

The same status and error reporting considerations apply to nontransparent as to transparent task-to-task communication. Refer to Section 6.6 for information on status and error reporting.

7.8 SYSTEM SERVICES CALL SUMMARY

The following subsections describe the VAX/VMS system services you can use for nontransparent intertask communication over the network. Each subsection describes the use of the call, its format, the arguments associated with the call, and the return status information. Appendix C lists the entire set of network system service error messages.

The following system services are not described in detail below, because their usage is the same whether or not they are used in a networking context. For a description of these system services, see the VAX/VMS System Services Reference Manual.

$CANCEL	Cancel I/O on Channel
$CREMBX	Create Mailbox and Assign Channel
$GETDVI	Get Device/Volume Information

Note that $GETDVI performs the same function as the Get I/O Channel Information ($GETCHN) system service. However, DIGITAL recommends that you use the $GETDVI system service.

7.8.1 $ASSIGN (I/O Channel Assignment)

The $ASSIGN system service assigns a channel to refer to a logical link. You use this channel in all $QIO calls that communicate with a remote task. In addition, you can use the $ASSIGN system service call to associate a mailbox with the channel.

Format

```
$ASSIGN devnam ,chan ,[acmode] ,[mbxnam]
```

Arguments

devnam	address of a quadword descriptor of a character string containing the string _NET: or a logical name for _NET:.
chan	address of a word that will receive the assigned channel number.
acmode	access mode to be associated with this channel. The most privileged access mode used is the access mode of the caller. You can perform I/O operations on the

	channel only from equal or more privileged access modes.
mbxnam	address of a character string descriptor for the physical name of the mailbox to be associated with the channel. This mailbox remains associated with the channel until the channel is deassigned ($DASSGN).

Return Status

SS$_NORMAL	The service completed successfully.
SS$_INSFMEM	There is not enough system dynamic memory to complete the request.
SS$_NOPRIV	The issuing task does not have the required privileges to create a logical link to the designated target.
SS$_NOSUCHDEV	The network device driver is not loaded (for example, the DECnet-VAX software is not running currently on the local node).

7.8.2 $QIO (Requesting a Logical Link Connection)

The $QIO system service with a function code of IO$_ACCESS requests a logical link connection to a target task. You can send 1 to 16 bytes of optional data to the target task at the same time that you issue the $QIO system service.

Format

```
$QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,p2
```

Arguments

efn	number of the event flag to be set at request completion.
chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_ACCESS
iosb	address of a quadword I/O status block that is to receive the completion status.
astadr	entry point address of an AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	AST parameter to be passed to the AST completion routine.
p1	not used (omit the argument).
p2	address of a quadword descriptor of the NCB (see Section 7.3.1). Both the descriptor and the NCB must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_CONNECFAIL	The connection to a network object timed out or failed.
SS$_DEVOFFLINE	The physical link is shutting down.
SS$_FILALRACC	A logical link is already accessed on the channel (that is, a previous connect on the channel).
SS$_INSFMEM	There is not enough system dynamic memory to complete the request.
SS$_INVLOGIN	The access control information was found to be invalid at the remote node.
SS$_IVDEVNAM	The NCB has an invalid format or content.
SS$_LINKEXIT	The network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOLINKS	No logical links are available. The maximum number of logical links as set for the executor MAXIMUM LINKS parameter was exceeded.
SS$_NOPRIV	The issuing task does not have the required privileges to create a logical link to the designated target.
SS$_NOSUCHNODE	The specified node is unknown.
SS$_NOSUCHOBJ	The network object number is unknown at the remote node; or for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	The remote node could not recognize the login information supplied with the connection request.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.
SS$_REJECT	The network object rejected the connection.
SS$_REMRSRC	The link could not be established because system resources at the remote node were insufficient.
SS$_SHUT	The local or remote node is no longer accepting connections.
SS$_THIRDPARTY	The logical link was terminated by a third party (for example, the System Manager).
SS$_TOOMUCHDATA	The task specified too much optional or interrupt data.
SS$_UNREACHABLE	The remote node is currently unreachable.

7.8.3 $QIO (Accepting a Logical Link Connection Request)

The $QIO system service with a function code of IO$_ACCESS accepts a logical link connection request from a source task. You can send 1 to 16 bytes of optional data to the source task at the same time that you issue the $QIO system service.

Format

```
    $QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,p2
```

Arguments

efn	number of the event flag to be set at request completion.
chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_ACCESS
iosb	address of a quadword I/O status block that is to receive the completion status.
astadr	entry point address of an AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	AST parameter to be passed to the AST completion routine.
p1	not used (omit the argument).
p2	address of a quadword descriptor of the NCB (see Section 7.3.1). Both the descriptor and the NCB must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_DEVALLOC	The process cannot access the logical link specified in the NCB because that link is intended for another process.
SS$_EXQUOTA	The process does not have sufficient quota to complete the request.
SS$_INSFMEM	There is not enough system dynamic memory to complete the request.
SS$_IVDEVNAM	The NCB has an invalid format or content.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited.
SS$_NOSUCHNODE	The specified node is unknown.

SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).
SS$_TIMEOUT	The connection request did not complete within the required time.
SS$_UNREACHABLE	The remote node is currently unreachable.

7.8.4 $QIO (Rejecting a Logical Link Connection Request)

The $QIO system service with a function code of IO$_ACCESS!IO$M_ABORT rejects a logical link connection request. You can send 1 to 16 bytes of optional data to the target task at the same time that you issue the $QIO system service.

Format

```
    $QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,p2
```

Arguments

efn	number of the event flag to be set at request completion.
chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_ACCESS!IO$M_ABORT
iosb	address of a quadword I/O status block that is to receive the completion status.
astadr	entry point address of an AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	AST parameter to be passed to the AST completion routine.
p1	not used (omit the argument).
p2	address of a quadword descriptor of the NCB (see Section 7.3.1). Both the descriptor and the NCB must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_DEVALLOC	The process cannot access the logical link specified in the NCB because that link is intended for another process.

SS$_EXQUOTA	The process does not have sufficient quota to complete the request.
SS$_IVDEVNAM	The NCB has an invalid format or content.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited.
SS$_NOSUCHNODE	The specified node is unknown.
SS$_TIMEOUT	The connection request did not complete within the required time.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).
SS$_UNREACHABLE	The remote node is currently unreachable.

7.8.5 $QIO (Sending a Message to a Target Task)

The $QIO system service with a function code of IO$_WRITEVBLK sends a message to a target task. Refer to Section 6.7.2 for the format of this call, its arguments, and possible return status codes.

7.8.6 $QIO (Receiving a Message from a Target Task)

The $QIO system service with a function code of IO$_READVBLK receives a message from a target task. Refer to Section 6.7.3 for the format of this call, its arguments, and possible return status codes.

7.8.7 $QIO (Sending an Interrupt Message to a Target Task)

The $QIO system service with a function code of IO$_WRITEVBLK!IO$M_INTERRUPT sends a 1- to 16-byte interrupt message to a target task. If the remote node is a VAX/VMS system, the message is placed in the mailbox associated with the target task.

Format

```
$QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,p1 ,p2
```

Arguments

efn	number of the event flag to be set at event completion.

chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_WRITEVBLK!IO$M_INTERRUPT
iosb	address of a quadword I/O status block that will receive the completion status.
astadr	entry point address of the AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	the AST parameter to be passed to the AST completion routine.
p1	buffer address.
p2	buffer length (1 to 16 bytes).

Return Status

SS$_NORMAL	The service completed successfully.
SS$_ABORT	The I/O request has been aborted by a $DASSGN or $CANCEL.
SS$_FILNOTACC	No logical link is associated with the channel.
SS$_INSFMEM	Enough memory to buffer the message could not be allocated.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited.
SS$_NOSOLICIT	DECnet could not accept an interrupt message at this time.
SS$_TOOMUCHDATA	The task specified too much interrupt data.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).

7.8.8 $QIO (Synchronously Disconnecting a Logical Link)

The $QIO system service with a function code of IO$_DEACCESS!IO$_SYNCH synchronously disconnects the logical link. All pending messages are transmitted to the remote node before the link is disconnected.

You can send 1 to 16 bytes of optional data to the task from which you are disconnecting at the same time you issue this $QIO system service.

Format

```
    $QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,[p2]
```

Arguments

efn	number of the event flag to be set at event completion.
chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_DEACCESS!IO$M_SYNCH
iosb	address of a quadword I/O status block that will receive the completion status.
astadr	entry point address of the AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	the AST parameter to be passed to the AST completion routine.
p1	not used (omit the argument).
p2	address of a descriptor of a counted ASCII string of optional user data. Both the string and its descriptor must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_FILNOTACC	No logical link is associated with the channel.

7.8.9 $QIO (Aborting a Logical Link)

The $QIO system service with a function code of IO$_DEACCESS!IO$_ABORT terminates the logical link. Note, however, that the DEACCESS function completes only after all pending I/O operations complete, even though you specify IO$_ABORT. First, issue the $CANCEL system service call to cancel I/O operations on the logical link and then issue this call to terminate the logical link.

You can send 1 to 16 bytes of optional data to the task from which you are disconnecting at the same time that you issue this $QIO system service call.

Format

```
    $QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,[p2]
```

Arguments

efn	number of the event flag to be set at event completion.

chan	address of a word containing the channel number associated with the logical link. Use the same channel number returned previously in the $ASSIGN call.
func	IO$_DEACCESS!IO$M_ABORT
iosb	address of a quadword I/O status block that will receive the completion status.
astadr	entry point address of the AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	the AST parameter to be passed to the AST completion routine.
pl	not used (omit the argument).
p2	address of a quadword descriptor of a counted string of optional user data. Both the string and its descriptor must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_FILNOTACC	No logical link is associated with the channel.

7.8.10 $QIO (Declaring a Network Name or Object Number)

The $QIO system service with a function code of IO$_ACPCONTROL assigns a network name or object number to the task, thereby making it eligible to process multiple inbound connection requests. You must associate a mailbox with the I/O channel. All inbound connection requests are placed in the mailbox associated with the channel over which this I/O function is issued. The SYSNAM privilege is required to declare a name or object number.

MACRO programmers should be aware that whenever a logical link is established, its device unit number (for example, 18 from _NET18:) should be obtained by using the $GETDVI system service, because unit numbers and not channel numbers appear in mailbox messages. Use this system service call where a single mailbox is being used for many logical links. The unit number could be used as a key into a data base which keeps track of multiple links.

Format

```
$QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,pl ,p2
```

Arguments

efn	number of the event flag to be set at event completion.
chan	a word containing the channel number associated with the logical link. Use the same channel number assigned previously in the $ASSIGN call.
func	IO$_ACPCONTROL
iosb	address of a quadword I/O status block that will receive the completion status.

astadr	entry point address of the AST routine that executes when the I/O operation completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.
astprm	the AST parameter to be passed to the AST completion routine.
p1	address of a quadword descriptor of a 5-byte block consisting of a function type (one byte) and a longword parameter. The function type is a symbol defined by the $NFBDEF macro in SYS$LIBRARY:LIB.MLB. The format of the 5-byte block for declaring a name is:

```
.BYTE NFB$C_DECLNAME
.LONG 0
```

The format of the 5-byte block for declaring an object number is:

```
.BYTE NFB$C_DECLOBJ
.LONG object-number
```

where the object number is a number reserved for customer use in the range of 128 to 255. This 5-byte buffer and its descriptor should be in read/write storage.

p2	address of a quadword descriptor of the network name (maximum of 12 characters). This is ignored for the DECLOBJ function. Both the name and its descriptor must be in read/write storage.

Return Status

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	One of the QIO parameters has an invalid value.
SS$_ILLCNTRFUNC	The control function is invalid.
SS$_NOMBX	A name or object number is being declared using a channel without an associated mailbox.
SS$_NOPRIV	The issuing process does not have the SYSNAM privilege.

7.8.11 $DASSGN (Terminating a Logical Link)

The $DASSGN system service terminates all pending operations to send and receive data, aborts the logical link immediately, and frees the channel associated with that link. Refer to Section 6.7.6 for the format of this call, it arguments, and possible return status codes.

7.9 PROGRAMMING EXAMPLE FOR NONTRANSPARENT COMMUNICATION

Example 7-1 illustrates the use of several of these system service calls for nontransparent task-to-task communication. CONNECT is a nontransparent MACRO source task on the local node that communicates with a nontransparent target task, DECLARNAM, on node DENVER. This example is similar to Example 6-1, except that here the source task uses an NCB and performs a nontransparent assign operation to establish communication with the target task. DECLARNAM is a nontransparent target task that has declared a name (that is, it is eligible to receive multiple inbound connection requests). In addition, it also uses a mailbox to receive network status notifications. Neither task performs useful functions. They are presented here only to illustrate various nontransparent functions.

Example 7-1: Nontransparent Task-to-Task Communication Using System Services

DECLARNAM

```
        .TITLE  DECLARNAM- NONTRANSPARENT EXAMPLE (WITH DECLARED NAME)
        .IDENT  /V1.0/
        .SBTTL  READONLY_DATA
        .PSECT  DECLARNAM$RDDATA        SHR,NOEXE,RD,NOWRT,BYTE

        $NFBDEF
        $DIBDEF

DEVDESC:.ASCID  /_NET:/                 ; Pseudo-device & descriptor
MAXMSG: .LONG   128                     ; Maximum message size
BUFQUO: .LONG   128                     ; Buffer quota
MBXDESC:.ASCID  /DECLARMBX/             ; Mailbox logname & descriptor
MAXLINKS=128                            ; Maximum # of logical links allowed
BUFFER_SIZE=64                          ; Size of input buffer to accept
                                                ;  messages
                .SBTTL  READWRITE_DATA
                .PSECT  DECLARNAM$RWDATA        SHR,NOEXE,RD,WRT,BYTE

        NAMEDESC:                               ; Declared name & descriptor
                .ASCID  /DECLAR/
        DCL_CHAN:
                .BLKW   1                       ; Word to receive declared name chan #
        DEV_CHAN:
                .BLKW   1                       ; Word to receive device channel #
        MBX_CHAN:
                .BLKW   1                       ; Word to receive mailbox channel #
        MBXMSG: .BLKB   128                     ; Mailbox buffer
        IOSB:   .BLKQ   1                       ; I/O status block
        AST_IOSB:                               ; I/O status block for AST
                .BLKQ   1
        NCBDESC:.LONG   100                     ; Network control block descriptor
                .LONG   NCB
        NCB:    .BLKB   100                     ; Network control block
        NFBDESC:.LONG   5                       ; Network function block descriptor
                .LONG   NFB
        NFB:    .BYTE   NFB$C_DECLNAME          ; Network function block
                .LONG   0
        PRILEN: .WORD   0                       ; Length of buffer for $GETCHAN info
        PRIBUF: .LONG   128                     ; Descriptor of $GETCHAN buffer
                .LONG   PBUF
        PBUF:   .BLKB   128                     ; Buffer to receive $GETCHAN info
        COUNT:  .BLKL   1                       ; Count of # of table entries
        CHAN_LIST:                              ; Channel # list
                .BLKW   MAXLINKS
        UNIT_LIST:                              ; Unit # list
                .BLKW   MAXLINKS
        IOSB_LIST:                              ; Read message I/O status block list
                .BLKQ   MAXLINKS
        BUFFERS:.BLKB   MAXLINKS * BUFFER_SIZE  ; Input buffers to put messages
```

```
BUF_ADR_LIST:                           ; List of pointers to input buffers
        OFFSET = 0
        .REPT   MAXLINKS
        .ADDRESS BUFFERS + OFFSET
        OFFSET = OFFSET + BUFFER_SIZE
        .ENDR

        .SBTTL  MAIN
;+
;
; This program demonstrates the use of a declared name to allow multiple
; inbound connection requests. It is included for illustrative purposes
; and is not intended to perform any useful work. In particular, nonnetwork
; code is kept to a minimum ( note for example, that  all errors result in
; program termination - an inappropriate procedure for most applications).
;
;-
        .PSECT  DECLARNAM$CODE          NOSHR,EXE,RD,NOWRT,BYTE

        .ENTRY  START,^M<>              ; Main entry point
;
; Create a temporary mailbox with the logical name DECLARMBX.
;
        $CREMBX_S -
                CHAN=W^MBX_CHAN,-       ; Adr of word to put mailbox channel
                MAXMSG=W^MAXMSG,-       ; Adr of longword with max message size
                BUFQUO=W^BUFQUO,-       ; Adr of longword with buffer quota
                LOGNAM=W^MBXDESC        ; Adr of descriptor of mbx logname

        BLBC    R0,EXITS                ; Error if LBC
;
; Assign a channel to a NET device and associate the mailbox with it.
;
        $ASSIGN_S -                     ; Issue $ASSIGN system service request
                DEVNAM=W^DEVDESC,-      ; Adr of descriptor of NET device
                CHAN=W^DCL_CHAN,-       ; Adr of word to put channel #
                MBXNAM=W^MBXDESC        ; Adr of descriptor of mbx logname

        BLBC    R0,EXITS                ; Error if LBC
;
; Declare a network name.
;
        $QIOW_S -                       ; Issue declare name request
                CHAN=W^DCL_CHAN,-       ; Use assigned channel
                FUNC=#IO$_ACPCONTROL,-  ; ACP QIO
                IOSB=W^IOSB,-           ; Adr of I/O status block
                P1=W^NFBDESC,-          ; Adr of NFB descriptor
                P2=#NAMEDESC            ; Adr of declared name descriptor

        BLBC    R0,EXITS                ; Error if LBC
        MOVZWL  W^IOSB,R0               ; Get the I/O status
        BLBC    R0,EXITS                ; Error if LBC
;
; Issue an asynchronous read to the mailbox.
;
        BSBW    READ_MBX                ; Set up mailbox read AST
;
; Now, go to sleep until someone writes to our mailbox.
;
        $HIBER_S                        ; zzzzzzz.......

EXITS:  $EXIT_S R0                      ; Exit with status

        .SBTTL  MAILBOX_AST
        .ENTRY  MAILBOX_AST,^M<>        ; Entry point for AST routine
```

```
;+
; AST routine to examine the mailbox message code
; and determine the appropriate action.
;-
        MOVZWL  W^AST_IOSB,R0           ; get async I/O completion status
        BLBC    R0,EXITS                ; Branch on failure

        CASEB   W^MBXMSG,#MSG$_ABORT,#MSG$_NETSHUT-MSG$_ABORT
DISP_TAB:
        .WORD   ABORT-DISP_TAB
        .WORD   CONFIRM-DISP_TAB
        .WORD   CONNECT-DISP_TAB
        .WORD   DISCON-DISP_TAB
        .WORD   EXIT-DISP_TAB
        .WORD   INTMSG-DISP_TAB
        .WORD   PATHLOST-DISP_TAB
        .WORD   PROTOCOL-DISP_TAB
        .WORD   REJECT-DISP_TAB
        .WORD   THIRDPARTY-DISP_TAB
        .WORD   TIMEOUT-DISP_TAB
        .WORD   NETSHUT-DISP_TAB
                                        ; Fall through on mbxmsg out of range
        BRB     EXITS                   ; Unknown mailbox message encountered
CONNECT:
        BSBW    CONNECT_ACCEPT          ; Go accept the connect request
        CMPW    #IO$_ACCESS!IO$M_ABORT,R4; Was the connect request rejected?
        BEQL    10$                     ; If rejected then do not issue the
                                        ;  read
        BSBW    READ_CHAN               ; Issue a read on the channel we
                                        ;  just confirmed
10$:    BSBB    READ_MBX                ; Requeue the mailbox read AST
        RET                             ; Return control to main program
DISCON:
ABORT:
EXIT:
PATHLOST:
PROTOCOL:
THIRDPARTY:
TIMEOUT:
        BSBW    DISCONNECT              ; Go disconnect the link
        BSBB    READ_MBX                ; Requeue the mailbox read AST
        RET                             ; Return control to main

NETSHUT:
        $WAKE_S                         ; The network is shutting down
        RET                             ; Wake the main program so that
                                        ;  it will exit

 INTMSG:                                ; Ignore interrupt messages for
                                        ;  this example
  REJECT:
  CONFIRM:
          BSBB    READ_MBX                ; Requeue the mailbox read AST
          RET                             ; Return control to main

          .SBTTL  READ_AST
          .ENTRY  READ_AST,^M<>           ; Entry point for AST routine
  ;+
  ; AST routine to check the completion status of the read and to process
  ; the message received.
  ;-
          MOVL    4(AP),R8                ; Get the index to the lists
          MOVQ    W^IOSB_LIST[R8],R0      ; Get the I/O completion status
          CMPW    #SS$_LINKABORT,R0       ; Did the link go away?
          BEQL    10$                     ; If EQL then ignore
          BLBC    R0,EXITS                ; If LBC then error
```

```
;
; OK, we have received a message and written it into the buffer pointed
; to by BUF_ADR_LIST[R1]. Useful code could be inserted here to process
; the message. When complete, we reissue the read on the channel and
; then go back to where we were interrupted.
;
        MOVW    W^CHAN_LIST[R8],R3      ; Get the channel number
        BSBB    READ_CHAN               ; Reissue the read
10$:    RET                             ; Return from AST routine

        .SBTTL  SUBROUTINES

;+
; Subroutine to issue an asynchronous read to the mailbox with an AST.
;-
READ_MBX:
        $QIO_S -                        ; Issue read with AST
                CHAN=W^MBX_CHAN,-       ; Use assigned mailbox channel
                FUNC=#IO$_READVBLK,-    ; Read virtual block
                IOSB=W^AST_IOSB,-       ; Address of I/O status block
                ASTADR=W^MAILBOX_AST,-  ; Address of AST routine
                P1=W^MBXMSG,-           ; Address of input buffer
                P2=W^MAXMSG             ; Length of input buffer
        BLBS    R0,10$                  ; Error if LBC
        BRW     EXITS                   ; Branch (failure)
10$:    RSB                             ; Return from subroutine

;+
; Subroutine to issue an asynchronous read to the channel with an AST.
;-
READ_CHAN:
        $QIO_S -                        ; Issue read with AST
                CHAN=R3,-               ; Use channel we just confirmed
                FUNC=#IO$_READVBLK,-    ; Read virtual block
                IOSB=W^IOSB_LIST[R8],-  ; Address of I/O status block
                ASTADR=W^READ_AST,-     ; Address of AST routine
                ASTPRM=R8,-             ; Store index as AST parameter
                P1=W^BUF_ADR_LIST[R8],- ; Address of input buffer
                P2=#BUFFER_SIZE         ; Length of input buffer
        BLBS    R0,10$                  ; Error if LBC
        BRW     EXITS                   ; Branch (failure)
10$:    RSB                             ; Return from subroutine

        .PAGE
;+
; Subroutine to accept the connect request and add the channel and unit
; numbers to the lists
;-
CONNECT_ACCEPT:
        MOVAB   W^MBXMSG+4,R9           ; Get adr of device name count
        MOVZBL  (R9)+,R8                ; Get byte count of device name string
        ADDL2   R8,R9                   ; Skip over the string
        MOVZBL  (R9)+,R8                ; Get byte count of info string
        MOVC3   R8,(R9),W^NCB           ; Put the NCB in adr NCB
        MOVL    R8,W^NCBDESC            ; Update the NCB descriptor
;
; Make sure we haven't reached maximum links.
;
        CMPL    #MAXLINKS,W^COUNT       ; Have we reached max # of links?
        BGTR    5$                      ; No? then go assign a channel to NET
        MOVW    W^DCL_CHAN,R3           ; Use the original channel
        MOVZWL  #IO$_ACCESS!IO$M_ABORT,R4;Setup with connect reject func code
        BRB     25$                     ; Go make the reject
```

```
5$:     $ASSIGN_S -                     ; Assign a channel to the task
                DEVNAM=W^DEVDESC,-      ; Adr of NET descriptor
                CHAN=W^DEV_CHAN,-       ; Channel #
                MBXNAM=W^MBXDESC        ; Associate the mailbox with new net dev
        BLBS    R0,10$                  ; If LBC then error
        BRW     EXITS                   ; Branch (failure)
;
; Now, get the channel and unit numbers and put them in their respective lists.
;
10$:    $GETCHN_S -                     ; Issue get chan system service
                CHAN=W^DEV_CHAN,-       ; Adr of word containing channel #
                PRILEN=W^PRILEN,-       ; Adr of word to put length returned
                PRIBUF=W^PRIBUF         ; Adr of descriptor of buffer
        BLBS    R0,19$                  ; If LBC then error
        BRW     EXITS                   ; Branch (failure)

19$:    CLRL    R8                      ; Initialize the index
20$:    TSTW    W^CHAN_LIST[R8]         ; Is it empty?
        BEQL    22$                     ; Branch if we found an empty slot
        AOBLEQ  W^COUNT,R8,20$          ; Inc the index and try again
        BRW     EXITS                   ; No empty slots?

22$:    MOVW    W^DEV_CHAN,W^CHAN_LIST[R8];Put the chan # in the chan list
        MOVW    W^PBUF+DIB$W_UNIT,W^UNIT_LIST[R8]; Put unit # in unit list
        INCL    W^COUNT                 ; Increment the # of entries
;
; Issue the connect confirm QIO.
;
        MOVW    W^DEV_CHAN,R3           ; Set up with adr of assigned chan
        MOVZWL  S^#IO$_ACCESS,R4        ; Set up with connect accept func code
25$:    $QIOW_S -                       ; Issue connect confirm/reject request
                CHAN=R3,-               ; Use assigned channel
                FUNC=R4,-               ; Request a logical link
                IOSB=W^IOSB,-           ; Address of I/O status block
                P2=#NCBDESC             ; Address of NCB descriptor

        BLBS    R0,30$                  ; If LBC then error
        BRW     EXITS                   ; Branch (failure)
30$:    MOVZWL  W^IOSB,R0               ; Get I/O completion status
        BLBS    R0,40$                  ; If LBC then error
        BRW     EXITS                   ; Branch (failure)
40$:    RSB                             ; Return from subroutine

;+
; Subroutine to disconnect a channel and remove its entry from the lists.
;-
DISCONNECT:
        MOVZWL  W^MBXMSG+2,R0           ; Get the unit #
        CLRL    R8                      ; Initialize the index
5$:     CMPW    W^UNIT_LIST[R8],R0      ; Locate the unit # in the list
        BEQL    10$                     ; If EQL then success
        AOBLEQ  W^COUNT,R8,5$           ; Inc the index and try again
10$:    MOVZWL  W^CHAN_LIST[R8],R9      ; OK, we've got the channel #
        $DASSGN_S -                     ; Deassign the channel
                CHAN=R9                 ; Channel #
        BLBS    R0,20$                  ; If LBC then error
        BRW     EXITS                   ; Branch (failure)
```

```
;
; OK, the channel has been deassigned. Remove the entries from the unit and
; channel lists.
;
20$:
        DECL    W^COUNT                 ; Decrement the # of entries
        CLRW    W^CHAN_LIST[R8]         ; Clear the chan list entry
        CLRW    W^UNIT_LIST[R8]         ; Clear the unit list entry

        RSB                             ; Return from subroutine

        .END    START                   ; Image transfer address
```

CONNECT

```
        .TITLE  CONNECT - ISSUE A CONNECT REQUEST TO DECLAR (DECLARED NAME)
        .SBTTL  READONLY_DATA
        .PSECT  CONNECT$RDDATA  SHR,NOEXE,RD,NOWRT,BYTE

DEVDESC:.ASCID  /_NET:/                 ; Pseudo-device & descriptor
MAXMSG: .LONG   64                      ; Largest size mailbox message allowed
BUFQUO: .LONG   64                      ; Mailbox quota

        .SBTTL  READWRITE_DATA
            .PSECT  CONNECT$RWDATA  SHR,NOEXE,RD,WRT,BYTE

    DEV_CHAN:
            .BLKW   1                       ; Word to receive device channel #
    MBX_CHAN:
            .BLKW   1                       ; Word to receive mailbox channel #
    MBXMSG: .BLKB   64                      ; Mailbox buffer
    IOSB:   .BLKQ   1                       ; I/O status block

    NCBDESC:.LONG   NCB_SIZE                ; Network connect block & descriptor
            .LONG   NCB
    NCB:    .ASCII  ?DENVER::?              ; Node-spec string
            .ASCII  ?"TASK=DECLAR/?         ; Task-spec string (declared name)
            .WORD   0                       ; Must be zero
            .ASCII  /"/                     ; End of NCB
    NCB_SIZE=.-NCB                          ; Size of NCB

            .SBTTL  MAIN

    ;+
    ; This program demonstrates how to make a connection request to a task
    ; which has declared a network name. It does not perform any useful work
    ; but does serve to illustrate DECnet nontransparent I/O.
    ;-
            .PSECT  CONNECT$CODE    NOSHR,EXE,RD,NOWRT,BYTE
            .ENTRY  START,^M<>

    ;+
    ; Use the run-time library routine, LIB$ASN_WTH_MBX to establish
    ; a communication path to DECnet software in preparation for
    ; nontransparent I/O operations.
    ;
    ; This routine will:
    ;
    ;       1. Create a temporary mailbox and assign a channel to it.
    ;
    ;       2. Assign a channel to _NET: and associate the temporary
    ;          mailbox with it.
    ;
    ; The mailbox can be used to obtain supplementary information from
    ; DECnet software about logical link operations.
    ;-
```

```
        PUSHAW  W^MBX_CHAN                   ; Adr to receive mailbox channel #
        PUSHAW  W^DEV_CHAN                   ; Adr to receive device channel #
        PUSHAL  W^BUFQUO                     ; Mailbox quota
        PUSHAL  W^MAXMSG                     ; Largest size mailbox message allowed
        PUSHAQ  W^DEVDESC                    ; Adr of device name descriptor
        CALLS   #5,LIB$ASN_WTH_MBX           ; Assign channel & associate mailbox
        BLBS    R0,10$                       ; If LBC then error
        BRW     EXITS                        ; Branch (failure)
;
; Request a logical link to the remote task.
;
10$:    $QIOW_S -                            ; Issue connect initiate request
                CHAN=W^DEV_CHAN,-            ; Use assigned channel
                FUNC=#IO$_ACCESS,-           ; Request a logical link
                IOSB=W^IOSB,-                ; Address of I/O status block
                P2=#NCBDESC                  ; Address of NCB descriptor

        BLBC    R0,EXITS                     ; If LBC then error
        MOVZWL  W^IOSB,R0                    ; Get I/O completion status
        BLBC    R0,EXITS                     ; If LBC then error

;+
; We now have a logical link with DECLARNAM. Useful code could be inserted
; here before continuing with the disconnect.
;
; OK, now disconnect the logical link by deassigning the channel.
;-

        $DASSGN_S -                          ; Issue the deassign request
                CHAN=W^DEV_CHAN              ; Adr of word containing channel #
        BLBC    R0,EXITS                     ; If LBC then error

EXITS:  $EXIT_S R0                           ; Exit with status to be displayed
                                                            ;  on error condition
                        .END    START
```

APPENDIX A

OBJECT TYPE CODE VALUES

Table A-1 defines valid object type code values and describes their use for task-to-task communication. All values are expressed in decimal.

Table A-1: Object Type Codes

Code	Object Type Mnemonic	Description
0	TASK	User program
1-16		Reserved for DIGITAL use
17	FAL	File Access Listener for remote file and record access
18	HLD	Host loader for RSX-11S down-line task loading requests
19	NML	Network Management Listener object
20-22		Reserved for DIGITAL use
23	REMACP	Network terminal handler (host side)
24		Reserved for DIGITAL use
25	MIRROR	Loopback mirror
26	EVL	Event receiver
27	MAIL	VAX/VMS mail facility
28		Reserved for DIGITAL use
29	PHONE	VAX/VMS phone facility
30-62		Reserved for DIGITAL use
63	DTR	DECnet Test Receiver object
64-127		Reserved for DIGITAL use
128-255		Reserved for customer use.

APPENDIX B

VAX-11 RMS CONTROL BLOCK USE

This appendix identifies which RMS control block fields are used by the network routines embedded in VAX-11 RMS when performing network file access and task-to-task operations. When both the source and target nodes are running VAX/VMS Version 3.0, most of the RMS control block fields are supported. The fields that are not used fall into two general categories: those that are ignored and those that are treated as an error condition if an unsupported value or bit option is specified. When the source node is running VAX/VMS in a heterogeneous network, certain fields of the FAB and RAB control blocks, and even entire XAB control blocks, may be ignored if they are not supported by the target node.

The general level of support provided by VAX-11 RMS and FAL for a given pair of nodes is determined dynamically by an exchange of information between them during a DAP setup sequence. The exact level of support, however, sometimes cannot be determined until the function is requested, as the remote FAL may return an error in response to a value or option it does not recognize or cannot process.

Tables B-1 through B-10 describe the support status of the fields in the FAB, RAB, NAM, and XAB blocks. The following key explains the support categories indicated in the tables:

KEY	Support Category For Field Value Or Bit Option
Yes	Supported; RMS at the source node will request that FAL at the target node perform the operation as specified.
RMS	Supported if the target node uses VAX-11 RMS or RMS-11 to perform its file operations; if not, the field is ignored (not used as an input or output).
VAX	Supported if the target node uses VAX-11 RMS to perform its file operations; if not, the field is ignored (not used as an input or output).
No	Not supported; the field is ignored (not used as an input or output).
Err	Not supported; an RMS$_SUPPORT error is returned.
NA	Not applicable; the field is not a user input or output.

Note that the comment "used locally" means that the bit option is input to VAX-11 RMS processing at the source node, but is not sent to FAL at the target node.

Table B-1: FAB (File Access Block)

Field	Name	Support	Comments
FAB$L_ALQ	Allocation quantity	Yes	
FAB$B_BID	Block identifier	NA	Static field
FAB$B_BKS	Bucket size	Yes	
FAB$B_BLN	Block length	NA	Static field
FAB$B_BLS	Block size	Yes	
FAB$L_CTX	User context	Yes	For user
FAB$W_DEQ	Default file extension quantity	Yes	
FAB$L_DEV	Device characteristics	Yes	Output only field*
FAB$L_DNA	Default file specification string address	Yes	
FAB$B_DNS	Default file specification string size	Yes	
FAB$B_FAC	File access options	---	Listed by subfield
FAB$V_BIO	Block I/O operations	Yes	
FAB$V_BRO	Record I/O operations	VAX	Used with RAB$V_BIO
FAB$V_DEL	$DELETE operations	RMS	
FAB$V_GET	$GET and $FIND operations	Yes	
FAB$V_PUT	$PUT operations	Yes	
FAB$V_TRN	$TRUNCATE operations	RMS	
FAB$V_UPD	$UPDATE operations	RMS	
FAB$L_FNA	File specification string address	Yes	
FAB$B_FNS	File specification string size	Yes	
FAB$L_FOP	File processing options	---	Listed by subfield
FAB$V_CBT	Contiguous best try	VAX	
FAB$V_CIF	Create if nonexistent	Yes	
FAB$V_CTG	Contiguous allocation	Yes	

*This reflects the actual characteristics of the target device, if the remote node uses VAX/VMS and the RMS operation is $CREATE or $OPEN. It is not valid for a $PARSE call.

(continued on next page)

Table B-1 (Cont.): FAB (File Access Block)

Field	Name	Support	Comments
FAB$V_DFW	Deferred write	No	
FAB$V_DLT	Delete file on close	Yes	
FAB$V_MXV	Maximize version number	RMS	
FAB$V_NAM	Use name block	Yes	
FAB$V_NEF	Do not position at end-of-file	VAX	
FAB$V_NFS	Non file structured	Err	
FAB$V_OFP	Output file parse	Yes	Used locally
FAB$V_POS	Current position	VAX	
FAB$V_RCK	Read check	Yes	
FAB$V_RWC	Rewind file on close	VAX	
FAB$V_RWO	Rewind file on open	VAX	
FAB$V_SCF	Submit command file	Yes	
FAB$V_SPL	Spool file to printer	Yes	
FAB$V_SQO	Sequential only (enter DAP file transfer mode)	Yes	See Section 4.5.3
FAB$V_SUP	Supersede existing file	Yes	
FAB$V_TEF	Truncate at end-of-file	VAX	
FAB$V_TMD	Temporary marked for delete	Yes	
FAB$V_TMP	Temporary file	Yes	
FAB$V_UFO	User file open	Err	
FAB$V_WCK	Write check	Yes	
FAB$B_FSZ	Fixed control area size	Yes	
FAB$W_GBC	Global Buffer Count	No	
FAB$W_IFI	Internal file identifier	NA	For internal use
FAB$L_MRN	Maximum record number	Yes	
FAB$L_MRS	Maximum record size	Yes	
FAB$L_NAM	Name block address	Yes	Most fields of NAM block are used

(continued on next page)

Table B-1 (Cont.): FAB (File Access Block)

Field	Name	Support	Comments
FAB$B_ORG	File organization	---	Listed by value
FAB$C_IDX	Indexed	Yes	
FAB$C_REL	Relative	Yes	
FAB$C_SEQ	Sequential	Yes	
FAB$B_RAT	Record attributes	---	Listed by subfield
FAB$V_BLK	Records do not cross block boundaries	Yes	
FAB$V_CR	Implied carriage control (LF <record> CR)	Yes	
FAB$V_FTN	FORTRAN carriage control	Yes	
FAB$V_PRN	Print file format	Yes	
FAB$B_RFM	Record format	---	Listed by value
FAB$C_FIX	Fixed length	Yes	
FAB$C_STM	Stream with LF, FF, VT and CRLF terminator set	Yes	
FAB$C_STMCR	Stream with CR record terminator	Err	
FAB$C_STMLF	Stream with LF record terminator	Err	
FAB$C_VAR	Variable length	Yes	
FAB$C_VFC	Variable length with fixed control	Yes	
FAB$C_UDF	Undefined	Yes	
FAB$B_RTV	Retrieval window size	No	
FAB$L_SDC	Spooling device characteristics	Yes	Same as DEV field
FAB$B_SHR	File sharing options	---	Listed by subfield
FAB$V_DEL	Allow other DELETEs	VAX	
FAB$V_GET	Allow other GETs	Yes	
FAB$V_MSE	Multistream access enabled	No	
FAB$V_NIL	Prohibit file sharing	VAX	
FAB$V_PUT	Allow other PUTs	Yes	

(continued on next page)

Table B-1 (Cont.): FAB (File Access Block)

Field	Name	Support	Comments
FAB$V_UPD	Allow other UPDATEs	VAX	
FAB$V_UPI	User-provided interlocking	VAX	For block I/O
FAB$L_STS	Completion status code	Yes	Also returned in R0
FAB$L_STV	Status value	Yes	Has DAP code when STS=RMS$_SUP, STS=RMS$_NET, or STS=RMS$_BUG_DAP
FAB$L_XAB	Extended attribute block address	Yes	**

**In general, if the remote FAL does not support an extended attribute block, the XAB is not used as input or output for the operation.

Table B-2: RAB (Record Access Block)

Field	Name	Support	Comments
RAB$B_BID	Block identifier	NA	Static field
RAB$L_BKT	Bucket code	Yes	
RAB$B_BLN	Block length	NA	Static field
RAB$L_CTX	User context	Yes	For user
RAB$L_FAB	File access block address	Yes	
RAB$W_ISI	Internal stream identifier	NA	For internal use
RAB$L_KBF	Key buffer address	Yes	
RAB$B_KRF	Key of reference	Yes	
RAB$B_KSZ	Key size	Yes	
RAB$B_MBC	Multiblock count	No	
RAB$B_MBF	Multibuffer count	No	
RAB$L_PBF	Prompt buffer address	No	
RAB$B_PSZ	Prompt buffer size	No	
RAB$B_RAC	Record access mode	---	Listed by value
RAB$C_KEY	Random access by key value	Yes	

(continued on next page)

Table B-2 (Cont.): RAB (Record Access Block)

Field	Name	Support	Comments
RAB$C_RFA	Random access by file address of record	Yes	
RAB$C_SEQ	Sequential access	Yes	
RAB$L_RBF	Record buffer address	Yes	
RAB$W_RFA	Record's file address	Yes	
RAB$L_RHB	Record header buffer	Yes	
RAB$L_ROP	Record processing options	---	Listed by subfield
RAB$V_ASY	Asynchronous I/O	Yes	Used locally
RAB$V_BIO	Change to block I/O	VAX	Used with FAB$V_BRO
RAB$V_CCO	Cancel control O	No	
RAB$V_CVT	Convert to uppercase	No	
RAB$V_EOF	Position at end-of-file	Yes	
RAB$V_FDL	Fast record delete	Yes	
RAB$V_LOC	Locate mode	No	
RAB$V_KGE	Key is greater than or equal to	Yes	
RAB$V_KGT	Key is greater than	Yes	
RAB$V_LIM	Test for key limit	Yes	
RAB$V_LOA	Load buckets via fill size	Yes	
RAB$V_NLK	Do not lock record	VAX	
RAB$V_NXR	Nonexistent record processing	VAX	
RAB$V_PMT	Prompt on read	No	
RAB$V_PTA	Purge type-ahead buffer	No	
RAB$V_RAH	Read ahead	No	
RAB$V_REA	Lock record for read, allowing other readers	VAX	
RAB$V_RLK	Lock record for write, allowing other readers	VAX	
RAB$V_RNE	Read no echo	No	
RAB$V_RNF	Read no filter	No	

(continued on next page)

Table B-2 (Cont.): RAB (Record Access Block)

Field	Name	Support	Comments
RAB$V_RRL	Read regardless of lock	VAX	
RAB$V_TMO	Enable timeout	No	
RAB$V_TPT	Truncate put	VAX	
RAB$V_UIF	Update if	RMS	
RAB$V_ULK	Enable manual record unlocking	VAX	
RAB$V_WAT	Wait until record unlocked	VAX	
RAB$V_WBH	Write behind	No	
RAB$W_RSZ	Record size	Yes	
RAB$L_STS	Completion status code	Yes	Also returned in R0
RAB$L_STV	Status value	Yes	Has DAP code when STS=RMS$_SUP, STS=RMS$_NET, or STS=RMS$_BUG_DAP
RAB$B_TMO	Timeout period	No	
RAB$L_UBF	User record area address	Yes	
RAB$W_USZ	User record area size	Yes	

Table B-3: NAM (Name Block)

Field	Name	Support	Comments
NAM$B_BID	Block identifier	NA	Static field
NAM$B_BLN	Block length	NA	Static field
NAM$B_DEV	Device string length	Yes	
NAM$L_DEV	Device string address	Yes	
NAM$W_DID	Directory identification	No	Zeroed on output
NAM$B_DIR	Directory string length	Yes	
NAM$L_DIR	Directory string address	Yes	
NAM$T_DVI	Device identification	No	Zeroed on output

(continued on next page)

Table B-3 (Cont.): NAM (Name Block)

Field	Name	Support	Comments
NAM$L_ESA	Expanded string area address	Yes	
NAM$B_ESL	Expanded string length	Yes	Output only field
NAM$B_ESS	Expanded string area size	Yes	
NAM$W_FID	File identification	No	Zeroed on output
NAM$L_FNB	File name status bits	Yes	Except for NAM$V_HIGHVER and NAM$V_LOWVER
NAM$B_NAME	File name string length	Yes	
NAM$L_NAME	File name string address	Yes	
NAM$B_NODE	Node name string length	Yes	
NAM$L_NODE	Node name string address	Yes	
NAM$L_RLF	Related file NAM block address	Yes	
NAM$L_RSA	Resultant string area address	Yes	*
NAM$B_RSL	Resultant string length	Yes	Output only field*
NAM$B_RSS	Resultant string area size	Yes	
NAM$B_TYPE	File type string length	Yes	
NAM$L_TYPE	File type string address	Yes	
NAM$B_VER	File version string length	Yes	
NAM$L_VER	File version string address	Yes	
NAM$L_WCC	Wild card context	NA	For internal use

*If the remote FAL does not support the return of a resultant name string, then a copy of the expanded name string is returned in this field.

Table B-4: XABALL (Allocation Control XAB)

Field	Name	Support	Comments
XAB$B_AID	Area identification number	Yes	
XAB$B_ALN	Alignment boundary type	--	Listed by value
XAB$C_CYL	Cylinder	VAX	
XAB$C_LBN	Logical block number	VAX	
XAB$C_RFI	Related file	No	
XAB$C_VBN	Virtual block number	VAX	
XAB$L_ALQ	Allocation quantity	Yes	
XAB$B_AOP	Allocation options	---	Listed by subfield
XAB$V_CBT	Contiguous best try	VAX	
XAB$V_CTG	Contiguous allocation	Yes	
XAB$V_HRD	Hard error	VAX	
XAB$V_ONC	On cylinder boundary	VAX	
XAB$B_BKZ	Bucket size	Yes	
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$W_DEQ	Default extension quantity	Yes	
XAB$L_LOC	Location	VAX	
XAB$L_NXT	Next XAB address	Yes	
XAB$W_VOL	Relative volume number	Yes	

Table B-5: XABDAT (Date and Time XAB)

Field	Name	Support	Comments
XAB$B_BLN	Block length	NA	Static field
XAB$Q_BDT	Backup date and time	Yes	
XAB$Q_CDT	Creation date and time	Yes	
XAB$B_COD	Type code	NA	Static field
XAB$Q_EDT	Expiration date and time	Yes	
XAB$L_NXT	Next XAB address	Yes	
XAB$Q_RDT	Revision date and time	Yes	
XAB$W_RVN	Revision number	Yes	

Table B-6: XABFHC (File Header Characteristics XAB)

Field	Name	Support	Comments
XAB$B_ATR	Record attributes	Yes	Output only field
XAB$B_BKZ	Bucket size	Yes	Output only field
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$W_DXQ	Default file extension quantity	Yes	Output only field
XAB$L_EBK	End-of-file block	Yes	Output only field
XAB$W_FFB	First free byte in end-of-file block	Yes	Output only field
XAB$W_GBC	Global buffer count	No	Output only field
XAB$L_HBK	Highest virtual block in file	Yes	Output only field
XAB$B_HSZ	Fixed length control header size	Yes	Output only field
XAB$W_LRL	Longest record length	Yes	Input for $CREATE
XAB$W_MRZ	Maximum record size	Yes	Output only field
XAB$L_NXT	Next XAB address	Yes	
XAB$B_RFO	File organization and record format	Yes	Output only field
XAB$L_SBN	Starting logical block number (if contiguous)	Yes	
XAB$W_VERLIMIT	Version limit for file	No	Output only field

Table B-7: XABKEY (Key Definition XAB)

Field	Name	Support	Comments
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$B_DAN	Data bucket area number	Yes	
XAB$B_DBS	Data bucket size	Yes	Output only field
XAB$W_DFL	Data bucket fill size	Yes	
XAB$B_DTP	Data type of the key	---	Listed by value
XAB$C_BN2	Unsigned 2-byte binary	Yes	
XAB$C_BN4	Unsigned 4-byte binary	Yes	
XAB$C_IN2	Signed 2-byte integer	Yes	
XAB$C_IN4	Signed 4-byte integer	Yes	
XAB$C_PAC	Packed decimal	Yes	
XAB$C_STG	String	Yes	
XAB$L_DVB	First data bucket start virtual block number	Yes	Output only field
XAB$B_FLG	Key options flag	---	Listed by subfield
XAB$V_CHG	Can be changed	Yes	Alternate key only
XAB$V_DAT_NCMPR	Do not compress data	No	
XAB$V_DUP	Can be duplicate	Yes	
XAB$V_IDX_NCMPR	Do not compress index	No	
XAB$V_KEY_NCMPR	Do not compress key	No	
XAB$V_NUL	Null key value	Yes	Alternate key only
XAB$B_IAN	Index bucket area number	Yes	
XAB$B_IBS	Index bucket size	Yes	Output only field
XAB$W_IFL	Index bucket file size	Yes	
XAB$L_KNM	Key name address	Yes	
XAB$B_LAN	Lowest level of index area number	Yes	
XAB$B_LVL	Level of root buckets	Yes	Output only field
XAB$W_MRL	Minimum record length	Yes	Output only field
XAB$B_NSG	Number of key segments	Yes	Output only field

(continued on next page)

Table B-7 (Cont.): XABKEY (Key Definition XAB)

Field	Name	Support	Comments
XAB$B_NUL	Null key value	Yes	
XAB$L_NXT	Next XAB address	Yes	
XAB$W_POS0 through XAB$W_POS7	Key position	Yes	
XAB$B_PROLOG	Prolog level	No	Primary key only
XAB$B_REF	Key of reference	Yes	
XAB$L_RVB	Root index bucket start virtual block number	Yes	Output only field
XAB$B_SIZ0 through XAB$B_SIZ7	Key size	Yes	
XAB$B_TKS	Total key size	Yes	Output only field

Table B-8: XABPRO (File Protection XAB)

Field	Name	Support	Comments
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$W_GRP	Group number of file owner	Yes	See XAB$L_UIC*
XAB$W_MBM	Member number of file owner	Yes	See XAB$L_UIC*
XAB$B_MTACC	Magnetic tape accessibility	No	
XAB$L_NXT	Next XAB address	Yes	
XAB$W_PRO	File protection	Yes	
XAB$L_UIC	User identification code (contains group and member number subfields)	Yes	*

*Zero is returned if the target node uses a non-VAX/VMS syntax to express the group and member UIC fields.

Table B-9: XABRDT (Revision Date and Time XAB)

Field	Name	Support	Comments
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$L_NXT	Next XAB address	Yes	
XAB$Q_RDT	Revision date and time	Yes	
XAB$W_RVN	Revision number	Yes	

Table B-10: XABSUM (Summary XAB)

Field	Name	Support	Comments
XAB$B_BLN	Block length	NA	Static field
XAB$B_COD	Type code	NA	Static field
XAB$B_NOA	Number of allocation areas defined for file	Yes	Output only field
XAB$B_NOK	Number of keys defined for file	Yes	Output only field
XAB$L_NXT	Next XAB address	Yes	
XAB$W_PVN	Prologue version number	Yes	Output only field

APPENDIX C

SUMMARY OF NETWORK SYSTEM SERVICE ERROR MESSAGES

Table C-1 describes the system service error messages for task-to-task communications.

Table C-1: System Services Error Message Summary

Message	Meaning
SS$_ABORT	The I/O request has been aborted by a $DASSGN or $CANCEL.
SS$_BADPARAM	One of the QIO parameters has an invalid value.
SS$_CANCEL	The I/O on this channel has been cancelled.
SS$_CONNECFAIL	The connection to a network object timed out or failed.
SS$_DATAOVERUN	More bytes were sent than could be received in the supplied buffer.
SS$_DEVALLOC	The process cannot access the logical link specified in the NCB because that link is intended for another process.
SS$_DEVOFFLINE	The physical link is shutting down.
SS$_EXQUOTA	The process does not have sufficient quota to complete the request. Sufficient FILLM and BYTLM quotas are required to request or confirm a logical link.
SS$_FILALRACC	A logical link is already accessed on the channel (that is, a previous connect on the channel).
SS$_FILNOTACC	No logical link is associated with the channel.
SS$_ILLCNTRFUNC	The control function is invalid.
SS$_INSFMEM	There is not enough system dynamic memory to complete the request.
SS$_IVCHAN	An invalid channel number was specified.

(continued on next page)

Table C-1 (Cont.): System Services Error Message Summary

Message	Meaning
SS$_INVLOGIN	The access control information was found to be invalid at the remote node.
SS$_IVDEVNAM	The NCB or task specifier has an invalid format or content.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task exited before confirming the logical link.
SS$_NOLINKS	No logical links are available. The maximum number of logical links as set for the executor MAXIMUM LINKS parameter was exceeded.
SS$_NOMBX	A name or object number is being declared using a channel without an associated mailbox.
SS$_NOPRIV	The issuing task does not have the required privileges to create a logical link to the designated target; or it does not have NETMBX and is assigning _NET:; or it does not have SYSNAM and is declaring a name or object number.
SS$_NORMAL	The service completed successfully.
SS$_NOSOLICIT	DECnet could not accept an interrupt message at this time.
SS$_NOSUCHDEV	The network device is not loaded (for example, the DECnet-VAX software is not running currently on the local node).
SS$_NOSUCHNODE	The specified node is unknown.
SS$_NOSUCHOBJ	The network object number is unknown at the remote node; or for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	The remote node could not recognize the login information supplied with the connection request.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.

(continued on next page)

Table C-1 (Cont.): System Services Error Message Summary

Message	Meaning
SS$_REJECT	The network object rejected the connection.
SS$_REMOTE	The service successfully completed. (A logical link was established with the target task.) This status applies only to an $ASSIGN call for a transparent link.
SS$_REMRSRC	The link could not be established due to insufficient system resources at the remote node.
SS$_SHUT	The local or remote node is no longer accepting connections.
SS$_THIRDPARTY	The logical link connection was terminated by a third party (for example, the System Manager).
SS$_TIMEOUT	The task did not respond to the connection request within the required time.
SS$_TOOMUCHDATA	The task specified too much optional or interrupt data.
SS$_UNREACHABLE	The remote node is currently unreachable.

APPENDIX D

MAILBOX MESSAGE TYPES

The $MSGDEF macro defines the mailbox messages described in Table D-1. This table defines the message type, its meaning, and any information that may accompany the message.

Table D-1: Mailbox Message Summary

Type	Meaning	Information
MSG$_ABORT	The logical link was aborted.	Optional data
MSG$_CONFIRM	The logical link was confirmed.	Optional data
MSG$_CONNECT	The task received an initial connection request.	NCB
MSG$_DISCON	The logical link was disconnected.	Optional data
MSG$_EXIT	The partner task exited without completing outstanding I/O operations.	None
MSG$_INTMSG	Interrupt message	Message
MSG$_NETSHUT	The network node is going into the "Shut" or "Off" state.	None
MSG$_PATHLOST	The partner is no longer accessible.	None
MSG$_PROTOCOL	There is a general NSP problem.	None
MSG$_REJECT	The logical link was rejected.	Optional data
MSG$_THIRDPARTY	A third party disconnected the logical link.	None
MSG$_TIMEOUT	The connection request timed out.	None

GLOSSARY

access control

The login control that a node exercises over inbound logical link connection requests to determine whether or not the link can be accepted.

cooperating tasks

Two tasks that communicate with each other in a task-to-task communication environment. In particular, cooperating tasks must agree on optional user data to be passed, how they will send and receive messages to ensure that there is one transmit for each receive, and which task will disconnect the link.

disconnect abort

Nontransparent tasks can deaccess a logical link via a disconnect abort operation without deassigning the channel. This form of disconnection indicates to the receiver that not all messages sent have necessarily been received.

handshaking sequence

The exchange of logical link connection information between two tasks. This exchange takes place to enable the successful completion of a logical link connection.

inbound connection

The term refers to the fact that a task receives logical link connection requests.

interrupt message

A user-generated message sent "outside" the normal exchange of data messages during nontransparent task-to-task communication. This usage of the term "interrupt" is contrary to the normal usage, which means to designate a software or hardware interrupt mechanism.

local node

The node at which you are located physically.

network connect block

A user-generated data structure used in a nontransparent task to identify a remote task and optionally send user data in calls to request, accept, or reject a logical link connection.

network object

The term refers to any task with a nonzero object type (for example, those programs such as FAL and NML that provide generic services across a network).

network status notifications

Notifications that provide information about the state of both logical and physical links over which two tasks communicate. A nontransparent task can use this information to take appropriate action under conditions such as third party disconnections and a partner's exiting before I/O completion.

network task

A nontransparent task that is able to process multiple inbound connection requests: that is, it has declared a network name or object number.

nonprivileged

In DECnet-VAX terminology, this term means no privileges other than NETMBX and TMPMBX, which are the minimum requirements for any network activity.

object type

A discrete identifier for either a task or DECnet service on a remote node. Object type identifiers can either be 0 plus a name (alternatively, TASK=name), or nonzero without a name (for example, 17= or FAL=).

outbound connection

The term refers to the fact that a task sends logical link connection requests.

privileged

In DECnet-VAX terminology, this term means any user privileges in addition to NETMBX and TMPMBX.

remote node

Any node other than the one at which you are located in the network. (When debugging programs or for operational symmetry, however, you can treat the local node like a remote node.)

source task

The task that initiates a logical link connection request in a task-to-task communication environment.

synchronous disconnect

The disconnect that occurs when a nontransparent task can issue a call to terminate I/O operations over a logical link without deassigning the channel. Thus, the task can use the channel for subsequent I/O operations with the same or a different remote task.

target task

The task that receives and processes a logical link connection request in a task-to-task communication environment.

task

In this manual, the term refers to an image running in the context of a process.

task specifier

Information provided to DECnet-VAX software so that it can complete a logical link connection to a remote task. This information includes the name of the remote node on which the target task runs and the name of the task itself.

INDEX

Access control, 2-1, 2-5, 5-7 to 5-8
  Configuration Data Base, 2-7
  defaulting to, 2-2, 2-6
  delimiters, 2-3
  explicit definition of, 2-2, 2-6
  for inbound connection requests, 2-5, 2-8
  for outbound connection requests, 2-5, 2-8
  implicit definition of, 2-2, 2-7
  logical-nodename, 2-2
  LOGINOUT image, 2-5, 5-8
  node name, 2-2
  null information, 2-6
  privileges, 2-6
  string format, 2-2
  User Authorization File (UAF), 2-5, 5-8
Account,
  nonprivileged, 2-7
  privileged, 2-7
ANALYZE/RMS_FILE command, 3-7
APPEND command, 3-8
ASSIGN command, 3-4 to 3-5
$ASSIGN system service, 6-2, 6-4
  format, 6-4, 7-9
  _NET:, 7-9
  nontransparent use of, 7-2
  transparent use of, 6-2

BACKUP command, 3-8

$CANCEL system service, 7-8
Channel, 5-7
  assigning for logical link, 5-7, 6-4, 7-9
  deassigning, 5-11, 6-3
  _NET:, 7-2
$CLOSE call, 4-9, 4-12
CLOSE command, 3-17
Code,
  object type, A-1
  system service status return, 6-3, 7-9
  VAX-11 RMS completion, 4-3
Command procedure file,
  See DCL command procedure
Configuration Data Base, 2-7
$CONNECT call, 4-9
CONVERT command, 3-9
COPY command, 3-8, 3-13 to 3-14
$CREATE call, 4-12
CREATE command, 3-10
$CREMBX system service, 7-3

DAP (Data Access Protocol), 4-1
$DASSGN system service, 5-11, 6-3, 6-7, 7-18
  format, 6-8
DCL command procedure, 2-4, 3-14, 3-18, 5-9
  example using SYS$NET, 3-18
  examples using lexical functions, 3-18
  for starting object, 2-4, 5-9
  lexical functions in, 3-15
  submitting remote, 3-14
DCL (DIGITAL Command Language) command, 1-1, 3-1
  command and file qualifiers, 3-1
  for accessing records, 3-16
  for command procedure submission, 3-14
  for file handling, 3-5
  for file operations, 3-2
  for lexical functions, 3-4
  for logical name operations, 3-2, 3-4
  for record access operations, 3-4
  summary, 3-2, 3-6
DEASSIGN command, 3-4 to 3-5
Declared name,
  See Task
Declared number,
  See Task
DEFINE command, 3-4 to 3-5
DELETE command, 3-10
Destination descriptor,
  See NCB (Network Connect Block)
DIFFERENCES command, 3-11
DIRECTORY command, 3-11
Disconnect, 5-11
  abort, 5-11, 7-8
  synchronous, 5-11
$DISCONNECT call, 4-9
Double colon delimiter, 2-1

DTR object code, A-1
DUMP/RECORDS command, 3-12

$ERASE call, 4-12
Error message,
  See Message
Error reporting, 4-3, 6-3, 7-9
  RMS completion codes, 4-3
  system service status, 6-3, 7-9
EVL object code, A-1

F$FILE_ATTRIBUTES lexical function, 3-15 to 3-16
F$PARSE lexical function, 3-16
F$SEARCH lexical function, 3-16
FAB (File Access Block), 4-9, B-2
FAL (File Access Listener), 4-1
  object code, A-1
File sharing over network, 4-13
File specification, 2-1
  angle brackets, 2-1
  examples, 2-5
  foreign-file-spec, 2-1, 2-3
  full file specification, 2-1, 2-3
  node name, 2-1
  node-spec, 2-1
  null character, 2-1
  quoted string, 2-1, 2-3
  restrictions on processing, 4-12
  space character, 2-1
  square brackets, 2-1
  subdirectories, 2-3
  tab character, 2-1
  task-spec-string, 2-1, 2-4
  wild card character, 2-3
File transfer throughput, 4-12
  DAP file transfer mode (FTM), 4-12
  DAP record access mode (RAM), 4-12
  FOP option, 4-12
FOP disposition options, 4-12
Foreign-file-spec,
  See File specification
Full file specification,
  See File specification

$GETDVI system service, 7-9

Handshaking sequence,
  See Logical link
HLD object code, A-1

Interrupt message,
  See Message

LEF (Local Event Flag Wait) state, 6-2
Lexical functions, 3-15
  use in command procedure files, 3-15
LIB$ASN_WTH_MBX library routine, 5-9, 7-3
Logical link, 2-4, 5-4 to 5-5, 5-7, 5-11, 6-2
  aborting, 5-5, 7-8
  assigning channel for, 6-2, 7-9
  completing connection of, 5-8, 6-2, 7-5, 7-12
  disconnecting, 5-5, 5-11, 7-8, 7-15
  handshaking sequence, 5-7
  rejecting a request, 7-13
  requests, 2-5, 5-4, 5-7 to 5-8, 6-2, 7-4 to 7-5, 7-10
  SYS$NET, 5-8
  terminating, 5-5, 5-11, 6-3, 6-7, 7-9
Logical name, 2-1, 2-3, 2-9
  as device name, 2-9
  as node name, 2-9
  DCL commands for, 3-4
  equivalence string, 2-9
  examples, 2-9 to 2-10
  in process logical name table, 2-9
  iterative translation, 2-10
  translation, 2-3, 2-9
  use of, 2-9
  use of underscore (_), 2-2, 2-11
Logical node name,
  See File specification
LOGINOUT image, 2-5, 5-8, 7-6

Magnetic tape file
    restriction, 4-14
MAIL object code, A-1
Mailbox, 5-5, 7-2 to 7-3, D-1
  creating ($CREMBX), 7-3
  message format, 7-3
  message types, D-1
MERGE command, 3-13
Message, 5-4 to 5-5, 5-10,
    6-5 to 6-6
  data, 5-10
  error, 3-19, C-1
  exchanging, 5-10, 6-3, 7-7
  interrupt, 5-4 to 5-5, 7-7
  mailbox, 5-5, 5-10, D-1
  network status, 5-5
  optional user data, 5-4 to
      5-5, 5-7, 7-1
MIRROR object code, A-1
$MSGDEF macro, D-1
Multiple inbound connects,
    5-4, 7-7, 7-17

NAM (Name Block), B-7
  restrictions, 4-11
NCB (Network Connect Block),
    5-7, 7-4
  destination descriptor, 7-5
NET:, 7-2, 7-9
Network command terminal, 1-6,
    2-11
  SET HOST command, 2-11
Network example, 1-4
Network name, 2-4
  declaring, 2-4, 7-6, 7-17
Network task,
  declaring, 5-4, 5-9, 7-6
Node name,
  See File specification
Node-spec,
  See File specification
Nonprivileged account, 2-7
Nonprivileged network
    operations, 2-7
Nonzero object,
  See Object
Numbered object,
  See Nonzero object

Object, 2-4, A-1
  examples, 2-5
  known, 2-4
  nonzero object, 2-4
  number, 7-6, 7-17
  task, 2-4

Object (Cont.)
  type, 2-4, 5-7, A-1
  zero object, 2-4
$OPEN call, 4-9, 4-12
OPEN command, 3-17
Optional user data,
  See Message

$PARSE call, 4-12
PHONE object code, A-1
PRINT/REMOTE command, 3-12 to
    3-13
Privilege,
  associated with objects, 2-8
Privileged,
  account, 2-7
  network operations, 2-7
PURGE command, 3-10

$QIO(IO$_ACCESS!IO$M_ABORT)
    system service, 7-7
  format, 7-13
$QIO(IO$_ACCESS) system
    service, 7-4, 7-7
  format, 7-10, 7-12
$QIO(IO$_ACPCONTROL) system
    service, 7-7
  format, 7-17
$QIO(IO$_DEACCESS!IO$M_ABORT)
    system service, 7-8 to 7-9
  format, 7-16
$QIO(IO$_DEACCESS!IO$M_SYNCH)
    system service,
  format, 7-15
$QIO(IO$_READVBLK) system
    service, 6-6, 7-14
  format, 6-6
$QIO(IO$_WRITEVBLK!
    IO$M_INTERRUPT) system
    service, 7-7
  format, 7-14
$QIO(IO$_WRITEVBLK) system
    service, 6-5, 7-14
  format, 6-5
Quotation marks, 3-5
  triple set of, 3-5
Quoted string,
  See File specification

RAB (Record Access Block),
    4-9, B-5
READ command, 3-17
REMACP object code, A-1

Remote command terminal,
  See SET HOST and Network
    command terminal
Remote file access, 1-1
  FORTRAN program example, 4-8
  MACRO programming interface,
    4-1, 4-8
  using DCL, 3-1
  using VAX-11 RMS, 4-1
RMS control block field, B-1
  FAB, B-2
  NAM, B-7
  RAB, B-5
  XABALL, B-9
  XABDAT, B-10
  XABFHC, B-10
  XABKEY, B-11
  XABPRO, B-12
  XABRDT, B-13
  XABSUM, B-13
RMS (Record Management
    Services), 1-2, 4-1
  calls for block I/O
    processing, 4-11
  calls for file processing,
    4-10
  calls for file specification
    processing, 4-11
  calls for record processing,
    4-10
  completion codes, 4-3
  control blocks, B-1
  file system characteristics,
    4-3
  MACRO programming examples,
    4-14
  programming notes, 4-8, 4-11
  restrictions on use of, 4-2,
    4-11
  service call summary, 4-9
  service calls, 4-1, 4-8

$SEARCH call, 4-12
SEARCH command, 3-13
SET HOST command, 2-11
SHOW LOGICAL command, 3-5
SHOW TRANSLATION command, 3-5
SORT command, 3-13
Source task, 5-7
SUBMIT/REMOTE command, 3-14
Synchronous disconnect, 5-5,
    5-11, 7-8, 7-15
SYS$NET, 5-8, 6-2, 7-6
  use in command procedures,
    3-18
SYSNAM privilege, 7-7
System service call, 1-2,
    5-11, 6-1, 7-1
  error messages, C-1
  summary for nontransparent
    use, 7-1, 7-9
  summary for transparent use,
    6-1, 6-3

Target task, 5-7
Task, 1-2
  declaring for network, 5-4
  object, 2-4
  source, 5-10
  specification examples, 2-5
  specification string, 2-1,
    2-4
  target, 5-10, 6-5
  taskname, 2-4
Task-to-task communication,
    1-1, 2-1, 4-14, 5-1, 6-1,
    7-1
  nontransparent, 1-1, 5-1,
    5-4, 7-1
  nontransparent MACRO
    example, 7-18
  transparent, 1-1, 5-1, 6-1
  transparent FORTRAN example,
    5-2
  transparent MACRO example,
    6-8
$TRNLOG system service call,
    5-9
TYPE command, 3-15

UAF (User Authorization File),
    2-5, 5-8
Underscore character (_), 2-2
User Authorization File,
  See UAF

VAX-11 RMS,
  see RMS (Record Management
    Services),

Wild card character (*), 2-3
WRITE command, 3-17

XABALL (Allocation Control XAB), B-9
XABDAT (Date and Time XAB), B-10
XABFHC (File Header Characteristics XAB), B-10
XABKEY (Key Definition XAB), B-11
XABPRO (File Protection XAB), B-12
XABRDT (Revision Date and Time XAB), B-13
XABSUM (Summary XAB), B-13

Zero object,
  See Object