DECnet/E
Programmer's Reference Manual

Order No. AA-HJ94A-TC

January 1987

This manual describes the programming facilities available to users of DECnet/E V4.0, and is designed for experienced BASIC-PLUS, BASIC-PLUS-2, COBOL, FORTRAN, and MACRO programmers.

OPERATING SYSTEM AND VERSION:	RSTS/E	V9.3
SOFTWARE VERSION:	DECnet/E	V4.0

digital equipment corporation, maynard, massachusetts

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

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

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

Copyright © 1987 by Digital Equipment Corporation.
All rights reserved.

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

digital™	DECUS	Rainbow
	DECwriter	ReGIS
DATATRIEVE	DIBOL	RSTS
DEC	FMS-11	RSX
DECmail	LA	RT
DECmate	MASSBUS	UNIBUS
DECnet	PDP	VAX
DECsystem-10	P/OS	VMS
DECSYSTEM-20	Professional	VT
DECtape	Q-Bus	Work Processor

CONTENTS

PREFACE

PART I GENERAL INFORMATION

CHAPTER 1 Introduction

Basic Terms and Concepts 1-1
DECnet Networks 1-3
The DIGITAL Network Architecture 1-3
Network Routing 1-4
Overview of DECnet/E Structure 1-5

CHAPTER 2 Logical Links

Creating a Logical Link 2-2
Advantages of Logical Links 2-6
  Efficient Line Usage 2-6
  Separate Data by Intended Use 2-8
  Interrupt Messages 2-9
  Flow Control 2-10
DECnet/E Logical Link Limitations 2-11

CHAPTER 3 Overview of DECnet/E Message Services

General Remarks 3-3
Registering with the Monitor: Declare Receiver Call 3-4
  Declaring Identity 3-5
  Declaring Intended Usage 3-6
  DECnet/E Use of the Declare Receiver Call . . . 3-6
Remove Receiver Call 3-7
Get Local Node Parameters Call 3-7
Log User Event Call 3-8
Send/Receive Local Data Message 3-8
Send Local Data with Privilege Mask 3-8
Send/Receive Connect Initiate Message 3-9
Send/Receive Connect Confirm Message 3-11
Send/Receive Connect Reject Message 3-12
Send/Receive Network Data Message 3-12
Send/Receive Interrupt Message 3-13
Send/Receive Link Service Message 3-14
Send/Receive Disconnect Message 3-14
Send/Receive Link Abort Message 3-15

CHAPTER 4 Network Addressing and Flow Control

Network Addressing 4-1

Declaring Network Names and Object Type Codes . 4-2
Handling of Incoming Connect Requests 4-2
Incoming Connect Request Addressed to Object Type Code -- CI(n,null) 4-3
Incoming Connect Request Addressed to Name -- CI(0,name) 4-4
Multiple Copies 4-4
Multiple Copies Initiated by Terminal Users . 4-4
Multiple Copies Executed under PBS 4-5
Multiple Copies Initiated Through Automatic Start-Up 4-5
Automatic Job Start-Up 4-5
Flow Control 4-7
Transmit Queue Management 4-8
Backpressure Flow Control 4-11
Program-Regulated Flow Control 4-12
Flow Control on Incoming Data 4-14
Flow Control on Outgoing Data 4-14
Interrupt Message Flow Control 4-17
Programming Hints for Flow Control 4-19

PART II PROGRAMMING INFORMATION

CHAPTER 5 Network Programming in BASIC-PLUS

Programming Background 5-1
General Format of SYS Calls 5-1
Buffer Allocation for Message Operations 5-2
Notation Used in This Chapter 5-3
Declare Receiver (Local and DECnet) 5-4
Remove Receiver (Local and DECnet) 5-16
Get Local Node Parameters (DECnet) 5-18
Log User Event (DECnet) 5-20
Send SYS Calls 5-26
Send Local Data Message (Local) 5-26
Send Local Data Message with Privilege Mask (Local) 5-32
Send Connect Initiate Message (DECnet) 5-36
Send Connect Confirm Message (DECnet) 5-50
Send Connect Reject Message (DECnet) 5-56
Send Network Data Message (DECnet) 5-61
Send Interrupt Message (DECnet) 5-68
Send Link Service Message (DECnet) 5-74
Send Disconnect Message (DECnet) 5-80
Send Link Abort Message 5-85
Receive 5-90
Format of Received Local Data Message 5-99
Format of Received Local Data Message with Privilege Mask 5-100
Format of Received Connect Initiate Message . 5-101
Format of Received Connect Confirm Message . . 5-109

Format of Received Connect Reject Message . . 5-111
Format of Received Network Data Message . . . 5-112
Format of Received Interrupt Message 5-114
Format of Received Link Service Message . . . 5-115
Format of Received Disconnect Message 5-119
Format of Received Link Abort Message 5-120

CHAPTER 6 Network Programming in BASIC-PLUS-2

Programming Background 6-1
Coding 6-1
Declare Receiver: MDCL (Local and DECnet) 6-5
Remove Receiver: MREM (Local and DECnet) 6-13
Get Local Node Parameters: NTLN (DECnet) 6-15
Log User Event: NTEV (DECnet) 6-18
Send Calls 6-23
Send Local Data Message: MSLD (Local) 6-23
Send Local Data Message with Privilege Mask: MSPR (Local) 6-28
Send Connect Initiate Message: NTCI (DECnet) . 6-32
Send Connect Confirm Message: NTCC (DECnet) . 6-44
Send Connect Reject Message: NTCR (DECnet) . . 6-49
Send Network Data Message: NTDM (DECnet) . . . 6-53
Send Interrupt Message: NTIN (DECnet) 6-59
Send Link Service Message: NTLS (DECnet) . . . 6-64
Send Disconnect Message: NTDI (DECnet) 6-69
Send Link Abort Message: NTLA (DECnet) 6-73
Receive: MRCV (Local and DECnet) 6-77
Format of Received Local Data Message 6-84
Format of Received Local Data Message with Privilege Mask 6-85
Format of Received Connect Initiate Message . 6-86
Format of Received Connect Confirm Message . . 6-94
Format of Received Connect Reject Message . . 6-96
Format of Received Network Data Message . . . 6-97
Format of Received Interrupt Message 6-99
Format of Received Link Service Message . . . 6-100
Format of Received Disconnect Message 6-104
Format of Received Link Abort Message 6-105

CHAPTER 7 Overview of the Concise and Extended Interfaces for COBOL

The Concise COBOL Interface 7-1
The Extended COBOL Interface 7-3
Comparison of the Interfaces 7-5

CHAPTER 8 Concise COBOL Interface Calls

Programming Background 8-1
Coding 8-1
Compiling and Linking 8-2

Connect Request Call: CNTCON 8-3
Connect Accept Call: CNTACP 8-7
Send Record Call: CNTSND 8-11
Receive Record Call: CNTRCV 8-14
Disconnect Call: CNTDIS 8-17

CHAPTER 9 Extended COBOL Interface Call Formats

Programming Background 9-1
Coding 9-1
Tips for Returned Status Information in Byte Form 9-2
Tips for Setting Bits 9-4
Tips for Testing Bits 9-5
Compiling and Linking 9-6
Declare Receiver: MDCL (Local and DECnet) 9-7
Remove Receiver: MREM (Local and DECnet) 9-15
Get Local Node Parameters: NTLN (DECnet) 9-17
Log User Event: NTEV (DECnet) 9-20
Send Calls 9-25
Send Local Data Message: MSLD (Local) 9-25
Send Local Data Message with Privilege Mask: MSPR (Local) 9-30
Send Connect Initiate Message: NTCI (DECnet) . 9-34
Send Connect Confirm Message: NTCC (DECnet) . 9-46
Send Connect Reject Message: NTCR (DECnet) . . 9-51
Send Network Data Message: NTDM (DECnet) . . . 9-55
Send Interrupt Message: NTIN (DECnet) 9-62
Send Link Service Message: NTLS (DECnet) . . . 9-67
Send Disconnect Message: NTDI (DECnet) 9-72
Send Link Abort Message: NTLA (DECnet) 9-76
Receive: MRCV (Local and DECnet) 9-79
Format of Received Local Data Message 9-87
Format of Received Local Data Message with Privilege Mask 9-88
Format of Received Connect Initiate Message . 9-89
Format of Received Connect Confirm Message . . 9-97
Format of Received Connect Reject Message . . 9-99
Format of Received Network Data Message . . . 9-100
Format of Received Interrupt Message 9-101
Format of Received Link Service Message . . . 9-102
Format of Received Disconnect Message 9-106
Format of Received Link Abort Message 9-107

CHAPTER 10 Network Programming in FORTRAN

Programming Background 10-1
Coding 10-1
Compiling 10-4
Linking 10-4
Declare Receiver: MDCL (Local and DECnet) . . . 10-5
Remove Receiver: MREM (Local and DECnet) 10-14

Get Local Node Parameters: NTLN (DECnet) 10-16
Log User Event: NTEV (DECnet) 10-19
Send Calls 10-23
Send Local Data Message: MSLD (Local) 10-23
Send Local Data Message with Privilege Mask: MSPR (Local) 10-28
Send Connect Initiate Message: NTCI (DECnet) . 10-32
Send Connect Confirm Message: NTCC (DECnet) . 10-44
Send Connect Reject Message: NTCR (DECnet) . . 10-49
Send Network Data Message: NTDM (DECnet) . . . 10-53
Send Interrupt Message: NTIN (DECnet) 10-59
Send Link Service Message: NTLS (DECnet) . . . 10-63
Send Disconnect Message: NTDI (DECnet) 10-68
Send Link Abort Message: NTLA (DECnet) 10-72
Receive: MRCV (Local and DECnet) 10-76
Format of Received Local Data Message 10-83
Format of Received Local Data Message with Privilege Mask 10-84
Format of Received Connect Initiate Message . 10-85
Format of Received Connect Confirm Message . . 10-93
Format of Received Connect Reject Message . . 10-95
Format of Received Network Data Message . . . 10-96
Format of Received Interrupt Message 10-98
Format of Received Link Service Message . . . 10-99
Format of Received Disconnect Message . . . 10-103
Format of Received Link Abort Message . . . 10-104

CHAPTER 11 Network Programming in MACRO

Programming Background 11-1
Coding 11-1
Assembling 11-6
Linking 11-7
Declare Receiver: MDCL (Local and DECnet) . . . 11-8
Remove Receiver: MREM (Local and DECnet) 11-17
Get Local Node Parameters: NTLN (DECnet) 11-20
Log User Event: NTEV (DECnet) 11-24
Send Calls 11-30
Send Local Data Message: MSLD (Local) 11-30
Send Local Data Message with Privilege Mask: MSPR (Local) 11-36
Send Connect Initiate Message: NTCI (DECnet) . 11-41
Send Connect Confirm Message: NTCC (DECnet) . 11-54
Send Connect Reject Message: NTCR (DECnet) . . 11-60
Send Network Data Message: NTDM (DECnet) . . . 11-65
Send Interrupt Message: NTIN (DECnet) 11-71
Send Link Service Message: NTLS (DECnet) . . . 11-76
Send Disconnect Message: NTDI (DECnet) 11-82
Send Link Abort Message: NTLA (DECnet) 11-87
Receive: MRCV (Local and DECnet) 11-91
Format of Received Local Data Message 11-99

Format of Received Local Data Message with Privilege Mask 11-101
Format of Received Connect Initiate Message 11-103
Format of Received Connect Confirm Message . 11-112
Format of Received Connect Reject Message . 11-115
Format of Received Network Data Message . . 11-117
Format of Received Interrupt Message 11-119
Format of Received Link Service Message . . 11-121
Format of Received Disconnect Message . . . 11-125
Format of Received Link Abort Message . . . 11-127

CHAPTER 12 FIRQB and XRB Layouts for MACRO Message Calls

Declare Receive 12-2
Remove Receiver 12-3
Get Local Node Parameters 12-4
Log User Event 12-6
Send Local Data Message 12-8
Send Local Data Message with Privilege Mask . . 12-10
Send Connect Initiate Message 12-12
Send Connect Confirm Message 12-14
Send Connect Reject Message 12-16
Send Network Data Message 12-18
Send Interrupt Message 12-20
Send Link Service Message 12-22
Send Disconnect Message 12-24
Send Link Abort Message 12-26
Receive Message 12-28

APPENDIX A Network Object Type Codes

APPENDIX B Reason Codes for Connect Reject and Link Abort

INDEX

FIGURES

1-1 Topology of a Multiarea DECnet Network 1-2
2-1 ECL Handles Multiplexing and Demultiplexing of Data over Physical Lines 2-2
2-2 Logical Link Connections 2-4
2-3 Logical Links Provide Line Efficiency 2-7
2-4 ULAs, RLAs, and LLAs for Logical Links 2-8
2-5 Data is Queued for Logical Links in Separate Streams 2-9
2-6 Queueing of Interrupt Messages for Logical Links 2-10
4-1 How Transmit Queue Flow Control Affects a DECnet/E Program 4-10

4-2 How Backpressure Flow Control Affects a DECnet/E Program as a Transmitter 4-12
4-3 How the Flow Control Option Selected by the Remote Program Affects a DECnet/E Program as a Transmitter 4-16
4-4 How Interrupt Message Flow Control Affects a DECnet/E Program as a Receiver 4-18
5-1 Format of Connect Data Block on Send Connect Initiate Message 5-40
5-2 Format of Received Connect Data Block 5-103
6-1 Format of Connect Data Block on Send Connect Initiate Message 6-35
6-2 Format of Received Connect Data Block 6-88
9-1 Format of Connect Data Block on a Send Connect Initiate Message 9-37
9-2 Format of Received Connect Data Block 9-91
10-1 Format of Connect Data Block on Send Connect Initiate Message 10-35
10-2 Format of Received Connect Data Block 10-87
11-1 Format of Connect Data Block on Send Connect Initiate Message 11-44
11-2 Format of Received Connect Data Block 11-106

TABLES

3-1 Summary of Calls for Interprogram Communication . 3-2
5-1 Types of Receiver Access 5-9
5-2 System Validation of Name, Object, and Access Parameters 5-10
5-3 Format of Connect Data Block on Send Connect Initiate Message 5-41
5-4 Sender Selection Summary 5-97
5-5 Format of Received Connect Data Block 5-103
6-1 Types of Receiver Access 6-7
6-2 System Validation of Name, Object, and Access Parameters 6-8
6-3 Format of Connect Data Block on Send Connect Initiate Message 6-36
6-4 Sender Selection Summary 6-79
6-5 Format of Received Connect Data Block 6-88
9-1 Types of Access Permitted 9-9
9-2 System Validation of Name, Object, and Access Arguments 9-11
9-3 Format of Connect Data Block on a Send Connect Initiate Message 9-38
9-4 Sender Selection Summary 9-82
9-5 Format of Received Connect Data Block 9-91
10-1 Types of Receiver Access 10-7
10-2 System Validation of Name, Object, and Access Parameters 10-9

10-3 Format of Connect Data Block on Send Connect Initiate Message 10-36
10-4 Sender Selection Summary 10-79
10-5 Format of Received Connect Data Block 10-87
11-1 Types of Access Permitted 11-11
11-2 System Validation of Name, Object, and Access Parameters 11-13
11-3 Format of Connect Data Block on Send Connect Initiate Message 11-45
11-4 Sender Selection Summary 11-94
11-5 Format of Received Connect Data Block 11-106
A-1 Network Object Type Codes A-1
B-1 Reason Codes B-1

Preface

Objectives

This manual describes the DECnet/E services available to programmers, permitting them to code programs that exchange data with other programs running in a DECnet network environment.

Audience

This manual is written for users who are experienced programmers. While you are not expected to be knowledgeable of network programming, this manual assumes you are completely familiar with one of the supported programming languages and the general mechanisms for interfacing with the RSTS/E operating system. The supported programming languages are:

- BASIC-PLUS
- BASIC-PLUS-2
- COBOL
- FORTRAN
- MACRO

Document Structure

This manual is divided into two main parts:

Part I, General Information, contains four chapters that present:

- A general discussion of the DECnet/E environment (Chapter 1)
- A tutorial discussion of the DECnet/E features available for sending and receiving messages (Chapters 2 through 4)

Part II, Programming Information, contains eight chapters that describe the call formats you can use to access those features from the supported programming languages:

- BASIC-PLUS and BASIC-PLUS-2 (Chapters 5 and 6)
- Concise and extended COBOL (Chapters 7, 8, and 9)

- FORTRAN IV (Chapter 10)
- MACRO-11 (Chapters 11 and 12)

Related Documents

Instructions on how to compile your programs are given in the individual programming language guides. The *RSTS/E Task Builder Reference Manual* describes TKB, the linker used for building RSTS/E tasks from compiled programs. The *RSTS/E RT11 Utilities Manual* describes the RT11-based utilities that you use while programming on a RSTS/E system.

This manual assumes you are familiar with the concepts and features available for interfacing to the RSTS/E System Monitor, as presented in the following manuals:

- *RSTS/E System User's Guide*
- *RSTS/E Programming Manual*
- *RSTS/E System Directives Manual*

To obtain a general understanding of the DECnet environment (internal and external) you are directed to the *Introduction to DECnet Phase IV*.

DECnet/E also provides utility programs that let a terminal user communicate with another terminal user, manipulate files across the network, and copy the contents of one storage device to a device on another system. The information needed to run and use these utilities is given in the companion manual, *DECnet/E Utilities Guide*.

Two other companion manuals tell how to install, control, and manage a DECnet/E system:

- *DECnet/E Network Installation and Update Guide*
- *DECnet/E System Manager's Guide*

The more advanced network programmer may want to refer to the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for information concerning various network management functions.

Conventions

This manual uses the following conventions:

--> The arrow means points to, or contains the address of, as when the stack pointer register points to the first item in the stack. For example:

```
SP --> item at top of stack
       item one word down from top of stack
```

( ) Parentheses mean the contents of the item that the parentheses surround. For example, the contents of the program counter would be represented as:

```
(PC)
```

[ ] Square brackets around an item indicate the optional parts of a command in format statements. For example:

```
QIO$ paraml [,param2]
```

{ } Braces around two or more items indicate that you must choose one of the enclosed items. For example:

```
{QIO$}
{QIOW$}
```

< > Angle brackets around two or more items tell the MACRO assembler that the items make up a list. For example:

```
GLOBAL<namel[,name2,...]>
```

PART I

GENERAL INFORMATION

Chapter 1

Introduction

This chapter gives an overview of DECnet/E. A DECnet network is one in which two or more computer systems are running some version of DECnet software. DECnet/E is a combination of hardware and software that extends the capabilities of a RSTS/E system running on a DIGITAL PDP-11 computer. With DECnet/E, programmers can develop and execute programs that exchange data with remote programs running on another DECnet system in a network. This manual describes the features available to network programmers.

Basic Terms and Concepts

The term network refers to two or more computer systems connected such that they can exchange information. The computer systems, called nodes, are connected by communications paths called physical links. You can establish physical links through normal (switched) telephone circuits, several varieties of leased (private) lines, coaxial cables, or satellite transmission facilities.

Figure 1-1 shows the topology of a multiarea DECnet network. A node is considered to be adjacent to another node if there is a physical link between them. A path is the route over which data travels from its source to its destination within the network. Path length is the distance from the source node to the destination node, measured in hops. A hop is equal to a physical line between two adjacent nodes. In Figure 1-1, the path length between nodes BANGOR and BOSTON is one hop and between DENVER and LONDON is three hops. The network diameter is the maximum path length between any two nodes.

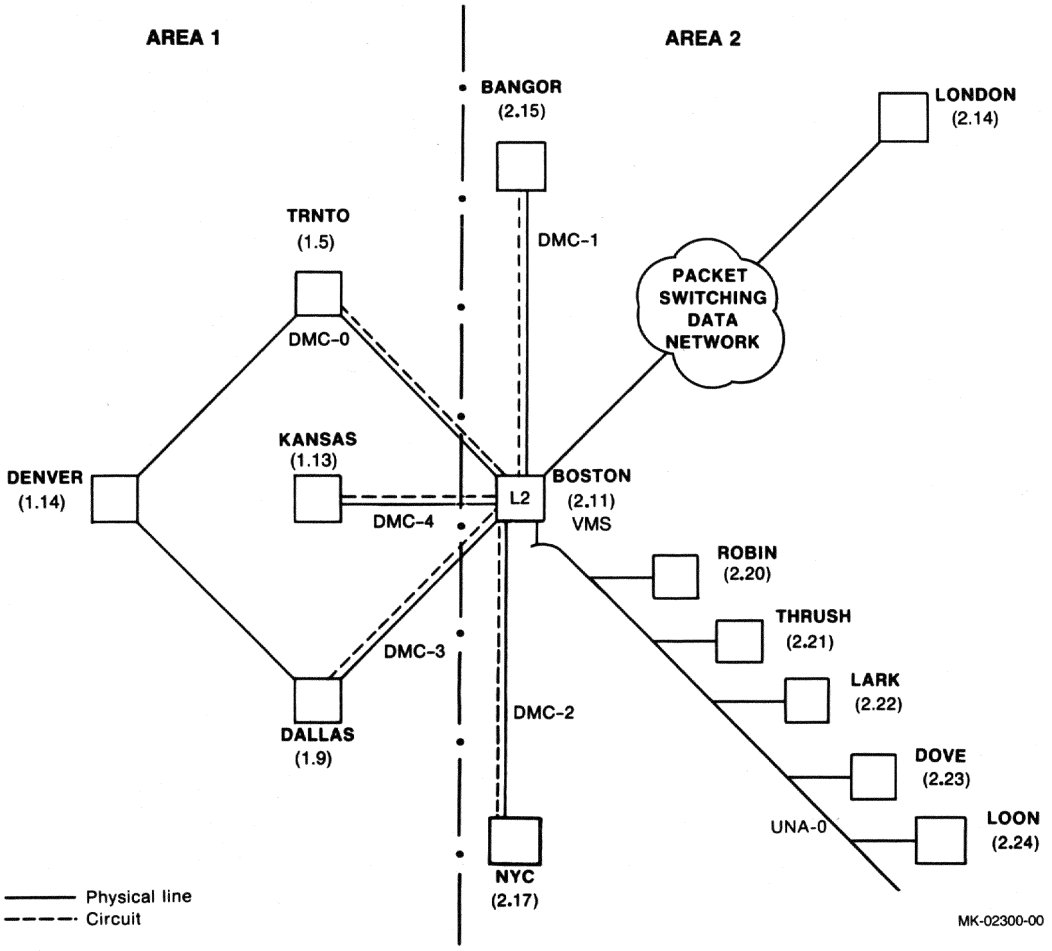


Figure 1-1: Topology of a Multiarea DECnet Network

The term remote program refers to any other program with which a program is communicating using the DECnet facilities. Remote programs do not need to be located on another node in the network. You can use DECnet/E to communicate with other programs at the local RSTS/E node. When developing network application programs, you can debug the whole system locally and later distribute it to remote nodes as required.

DECnet Networks

DECnet refers to the hardware and software that allows different DIGITAL systems to be connected to form networks. The design of DECnet allows each system to function as a separate entity, yet still allow access to resources that are distributed throughout the various systems in the network. A DECnet/E system provides all the capabilities of a RSTS/E system, and also provides the networking capabilities this manual describes. See the Preface for a list of the companion DECnet/E manuals.

DIGITAL bases all DECnet implementations on a logical structure called the DIGITAL Network Architecture (DNA). This architecture allows network implementations on the various DIGITAL systems to evolve (that is, to provide more and more capability as time goes on) within a basic common structure. The DNA structure ensures that implementations providing fewer features are always compatible with implementations providing more features.

At any given time, capabilities can differ from system to system. This document does not describe the DECnet implementations for all the different DIGITAL systems. However, to use DECnet/E, you must give some thought to the other DIGITAL systems in your network.

If your network consists entirely of DECnet/E Version 4.0 nodes, this document provides all the information necessary for you to use DECnet/E to send and receive messages in your programs. If your network contains other DECnet systems, check the DECnet documentation for those systems to be sure that the corresponding capabilities exist. For example, you can design network programs to be started automatically on RSTS/E systems. However, this feature is not available on all DECnet systems, so your design should not necessarily assume this capability.

See the *Introduction to DECnet Phase IV* for a brief overview of the functional capabilities of the various DECnet implementations.

The DIGITAL Network Architecture

DECnet/E is implemented according to a set of rules, or protocols, that govern the format, control, and sequencing of data exchange from the user program layer down through the physical link layer. These protocols are defined by the DIGITAL Network Architecture (DNA) -- a logical structure that provides the design model for all DECnet implementations.

DNA consists of several layers, each defining a distinct set of network functions and a set of rules for implementing those functions. Each DECnet implementation consists of software modules that perform these DNA-defined functions according to DNA-defined protocols. The relevant protocols for network programming are called the Network Services Protocol (NSP), the Routing protocol, and the Data Link layer protocols.

The NSP defines the rules for communication between programs in a network over paths called logical links. Logical links allow many different data streams to be multiplexed onto the physical link for transmission and separated at the receiving node for delivery to the appropriate program. See Chapter 2 for a detailed description.

The Routing protocol defines the rules for determining the actual physical path, or route, along which data travels to its destination.

The Data Link layer software controls the physical link. It provides correct sequencing of data and error control to ensure data integrity. This layer supports two protocols: DIGITAL Data Communications Message Protocol (DDCMP), and Ethernet. You use DDCMP to establish circuits between two adjacent DECnet nodes. Use the Ethernet protocol for Ethernet circuits over a coaxial cable based local area network (LAN).

Network Routing

DECnet/E Version 4.0 is a Phase IV DECnet product. See the *Introduction to DECnet Phase IV* for a discussion of the differences between Phase III and Phase IV DECnet products.

Phase IV features include:

- Backwards compatibility with Phase III.
- Increased network routing support allowing up to 63 areas (each area can have up to 1023 nodes) in a network.
- Enhanced network management capabilities.
- Expanded data link options to include support for DDCMP and Ethernet links.

Networks can consist of the following types of nodes:

- **Area routing nodes** -- Phase IV nodes which pass messages between areas.

- **Routing nodes** -- Phase III or IV nodes connected to multiple communications lines, supporting route-through capabilities.

- **Nonrouting nodes (end nodes)** -- Phase III or IV nodes connected to a single communications line.

Phase IV nodes can communicate with and are compatible with Phase III nodes. However, Phase III nodes do not acquire any new capabilities by being connected to Phase IV nodes. Phase III nodes can only communicate with nodes in their own area.

Overview of DECnet/E Structure

The DECnet/E features for network communication are part of the RSTS/E system monitor. The manner in which these features are accessed differs with the type of user.

- **User programs** -- Interface with DECnet/E through Session Control. Session Control is the operating specific interface to the DECnet/E implementation of the End Communications Layer (ECL). Although Session Control is not a separate entity or program, the term Session Control is used throughout this document to refer to the software that allows a user program to establish and exchange data over logical links. The RSTS/E BASIC programmer accesses Session Control services by use of the SYS() function. BASIC-PLUS-2 users can also use the SYS() function; however, DIGITAL recommends using calls to MACRO subroutines documented in this manual. COBOL, FORTRAN, and MACRO-11 programmers must use the MACRO subroutines to access the Session Control services.

- **Terminal users** -- Access certain network capabilities through various DECnet/E utility programs:

  - The utilities TLK (Talk) and LSN (Listen) let two terminal users type messages to each other.

  - The utilities NFT (Network File Transfer) and FAL (File Access Listener) work together to allow network file manipulation. NFT and FAL are implemented according to the Data Access Protocol (DAP) so terminal users can exchange files with other DECnet systems having the same capabilities.

- The NET utility lets the terminal user log in to and use a remote RSTS/E system as though the local terminal were directly connected to the remote system.

- NETCPY copies entire devices between RSTS/E nodes.

See the *DECnet/E Utilities Guide* for details on these utility programs.

o The **system manager** -- Uses the features provided by the Network Control Program (NCP) to monitor and control a node running in a network. See the *DECnet/E System Manager's Guide* for details on running NCP.

Chapter 2

Logical Links

This chapter introduces the logical link -- the basic element of DECnet task-to-task communication. A logical link is a software path for the exchange of data between two programs running in a network.

This chapter and the two chapters that follow discuss various logical link parameters -- object name and type code, local and remote link address, flow control data request counts, and so on. You can observe these parameters on an operational node by issuing NCP's SHOW LINK command (for a single link) or the SHOW ACTIVE LINKS command (for all current links). These commands can be helpful when you debug a network program. See the *DECnet/E System Manager's Guide* for details on using NCP commands.

In a network of DECnet nodes, many programs use the same physical path to exchange data. Data being sent from one program to another over a logical link is interspersed (multiplexed) with data sent over other logical links, transmitted over the physical line, and separated (demultiplexed) at the receiving end. The End Communications Layer (ECL) at each node handles this multiplexing and demultiplexing, accepting outgoing data from local programs and formatting it for transmission by the communications hardware. ECL also accepts incoming data from remote nodes and separates it into individual logical link streams that are then delivered to Session Control, which finds the appropriate user program to handle the message (see Figure 2-1).

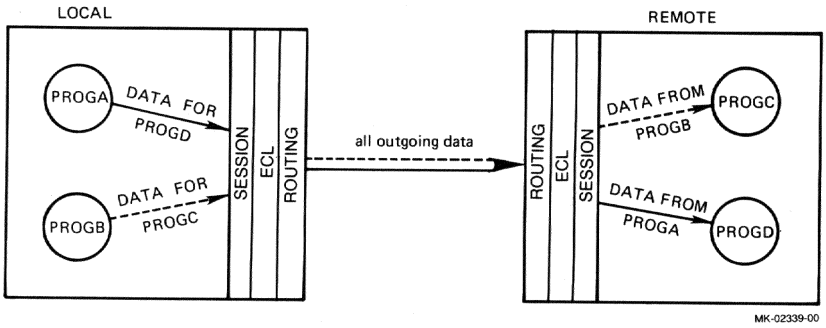


Figure 2-1: ECL Handles Multiplexing and Demultiplexing of Data over Physical Lines

Creating a Logical Link

Creating a logical link is a cooperative venture. Two programs must agree to communicate before a logical link is established. The procedure is basically the same for all implementations of DECnet. The program that initiates the request for a logical link connection is called the source. The other program is called the target. This distinction is made only during the connection sequence.

Note

Other network implementations make the source program the link master. The source must poll the target for data and only the source can disconnect the link. DECnet does not make this distinction. Once the connection is established, the terms source and target have no significance and both the local and remote programs have equal access to the logical link.

The request for a connection takes the form of a Connect Initiate Message that includes detailed information identifying the remote node and target program to which the connection is to be made. It also establishes an identifier for the link itself. In DECnet/E, the link identifier is called a User Link Address (ULA). A ULA is a number from 1 to 255 that is used by the source program to refer to the link during subsequent send and receive operations.

The Session Control at the source node does some initial checking of the request (for example, to determine if it recognizes the remote node). If the initial checking succeeds, Session Control requests ECL to send a connection request to the target node. The target node's software does its own checking to determine if the connection can be made. If the connection cannot be made, the target node rejects the connection. If the connection can be made, the remote node confirms the connection.

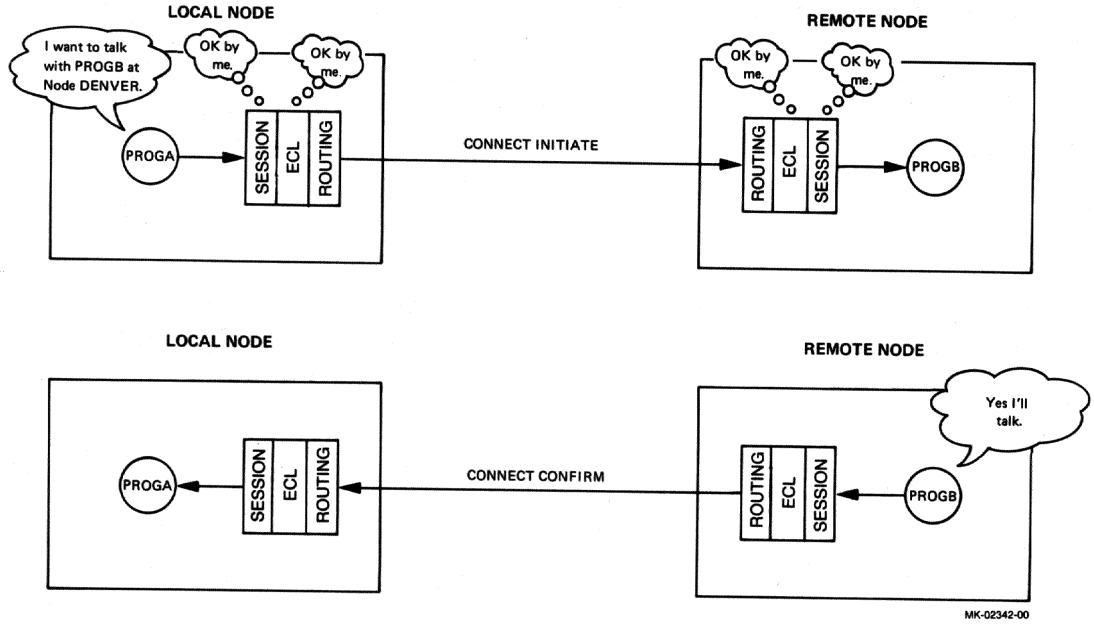
If the target program accepts the link, it selects its own link identifier by which it refers to the link. The two link identifiers are unrelated to one another. They can even take quite different forms if the programs are written in different languages or for different operating systems. If the remote node is a DECnet/E node, this identifier is again called a ULA and can range from 1 to 255. The number selected is completely independent of the selection made by the source program. The source and target ECLs relate the two identifiers to the same logical connection and keep the data exchanged over the link separate from data exchanged over other logical links.

The target program's confirmation or rejection of the link is passed back over the network to the source ECL, which in turn passes it to Session Control, which passes this information on to the source program. This confirmation or rejection takes the form of a Connect Confirm or Connect Reject Message that is queued for the source to receive and process.

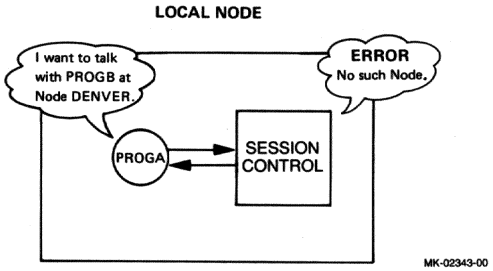
If the connection is confirmed after this initial exchange, the two programs can then send and receive data over the link, each referring to the link by the identifier it assigned during the connection sequence. Both programs can send data simultaneously, since the DECnet software provides logical full-duplex transmission regardless of the characteristics of the intervening network. Either program can break the logical link connection when it is no longer needed.

In Figure 2-2, diagram A shows the procedure for establishing a logical link connection. The target Session Control considers the link up (that is, it allows the target program to send data) when the source ECL acknowledges receipt of the Connect Confirm Message. The source Session Control considers the link up when it receives the Connect Confirm Message. Diagrams B through E of Figure 2-2 show the ways in which DECnet/E rejects a request for a link.

A. Remote program confirms logical link connection



B. Local SESSION CONTROL returns an error on connect request.



C. Local ECL returns an error on connect request.

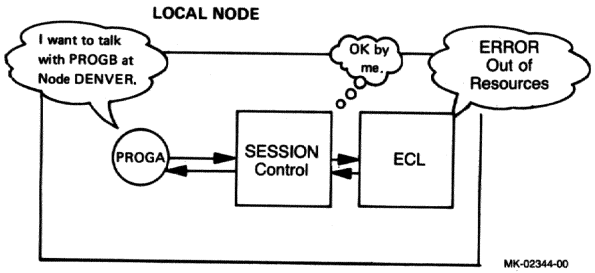
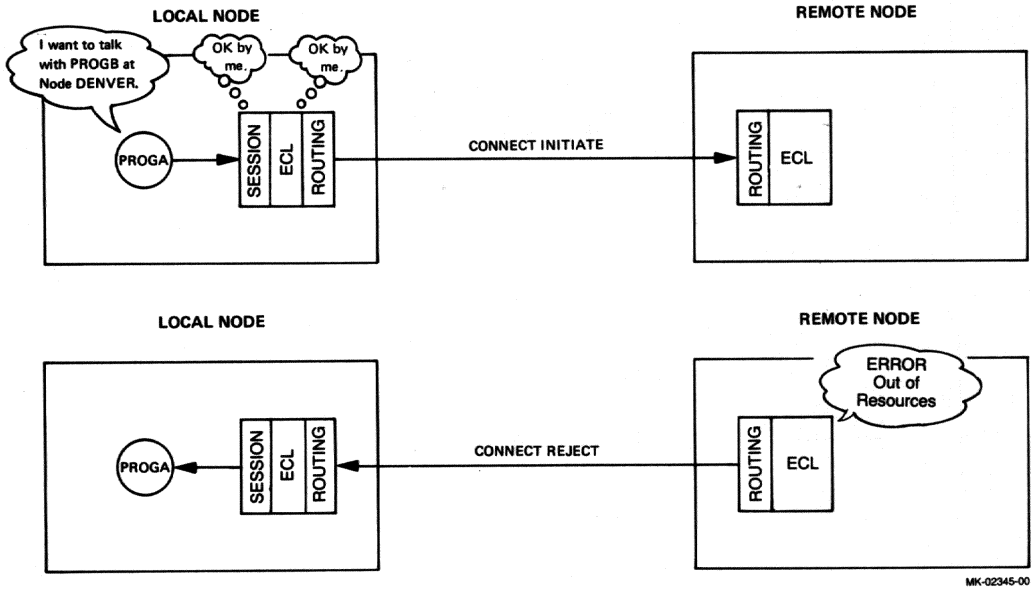


Figure 2-2: Logical Link Connections

D. Remote ECL rejects logical link connection.



E. Remote SESSION rejects logical link conection.

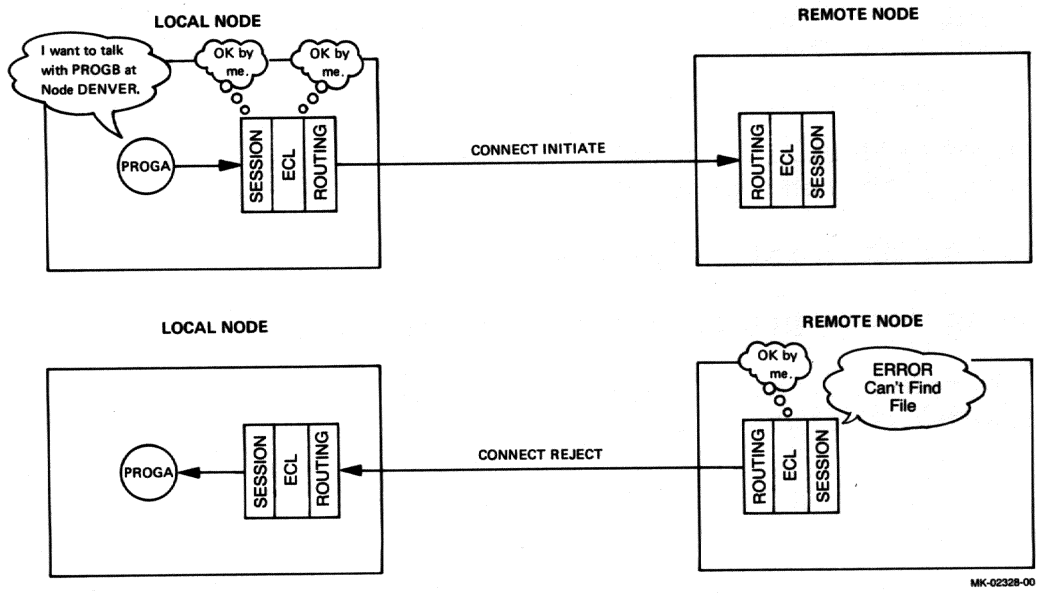


Figure 2-2: Logical Link Connections (Cont.)

F. Remote program rejects logical link connection.

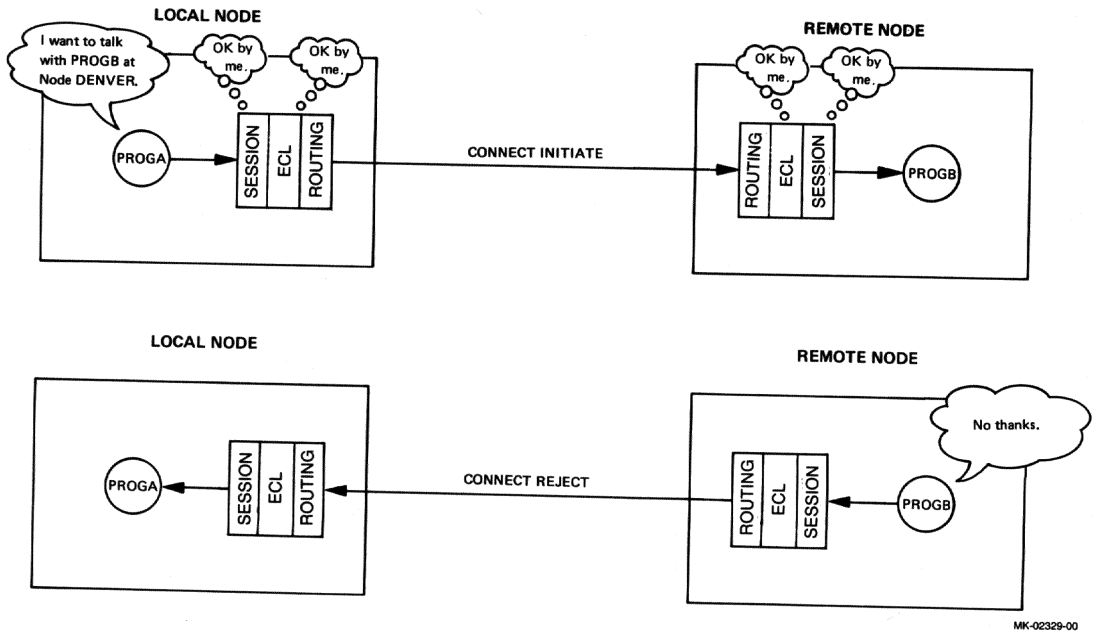


Figure 2-2: Logical Link Connections (Cont.)

Advantages of Logical Links

In addition to making it possible to sort out the many physical streams of data that pass over a single physical line, the logical link structure maintained by DECnet has several other advantages:

- Reduction of addressing data required for message transmission, thereby providing efficient line usage.
- Separation of data into separate message streams.
- Provision for high-priority Interrupt Messages.
- Ability to control the flow of data, thereby relieving possible network congestion.

Efficient Line Usage

Once established, a logical link simplifies the addressing necessary for the exchange of data between two programs, both at the user's level and in transmission of data across the network. As mentioned

previously, a link identifier is assigned by each program to identify an established logical link. This identifier is more convenient than the longer, detailed address used in identifying the remote node and program in the original Connect Initiate Message.

During a connection sequence, the source ECL and target ECL also establish a set of addresses that they use to identify the logical link. These addresses are called the Local Link Address (LLA) and the Remote Link Address (RLA), respectively. Each ECL refers to its own as the LLA and the other as the RLA.

While individual ULAs must be unique to the source program and to the target program, they are not required to be unique throughout the network, or even within the local node. Thus, the LLA and the RLA are required by ECL to separate the many distinct message streams that are multiplexed onto the communications lines. Since each ECL's LLA is unique to its own node, these ECL addresses reduce the amount of control and identifying information that must accompany the user data transferred across the link (see Figure 2-3).

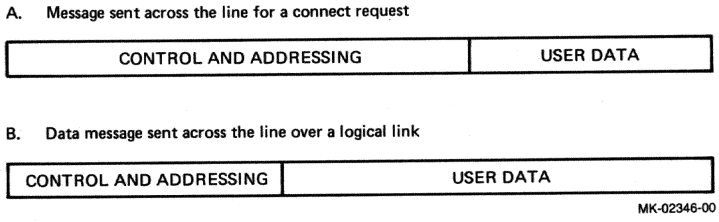


Figure 2-3: Logical Links Provide Line Efficiency

Figure 2-4 illustrates how these addresses work, showing two DECnet/E nodes with two established logical links. For example, Program A has established a logical link with Program D. Program A refers to the link with a ULA of 1. Program D refers to the link with a ULA of 23. Each ECL has established its own LLA. For Program A's ECL, the LLA is 123456. For Program D's ECL, the LLA is 024602. Each ECL keeps track of the other's LLA as a RLA.

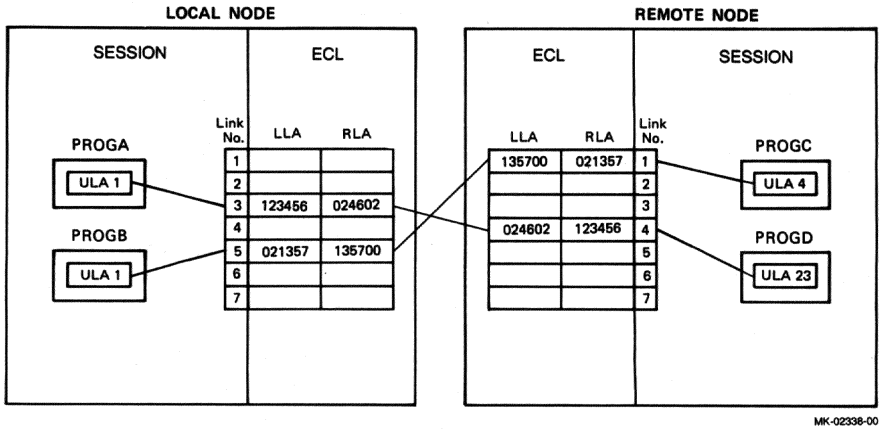


Figure 2-4: ULAs, RLAs, and LLAs for Logical Links

Separate Data by Intended Use

Logical links provide a means by which incoming messages can be separated into different message streams. A program can establish:

- Different logical links to communicate with different remote programs
- Several logical links with the same remote program to exchange data intended for different purposes

In either case, before a program can use the message services, it must first register this intention with Session Control. This involves declaring the name by which the program is to be known to other remote programs in the network (that is, its identity), as well as any limits or restrictions that affect the way in which the program can use the message services.

When a program declares its intent to use the network message services, Session Control sets up a receive queue to hold pending messages. Thereafter, all messages received for the program are placed in the queue. Messages from different logical links are interspersed in the queue but the receiving program can retrieve either the first message in the queue or the first message from a particular logical link.

Figure 2-5 shows the concept of separate message streams for separate logical links.

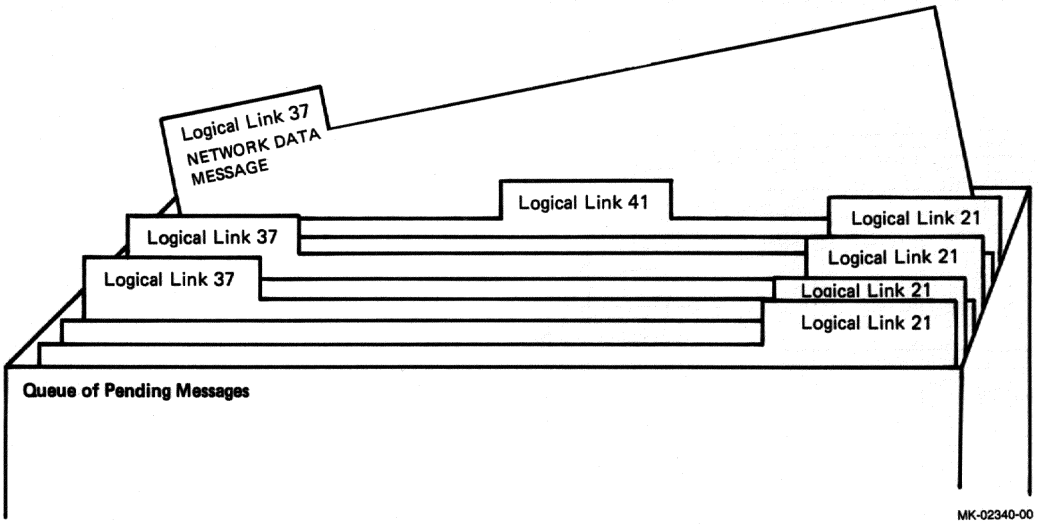


Figure 2-5: Data is Queued for Logical Links in Separate Streams

Interrupt Messages

Special high-priority Interrupt Messages can be sent over logical links in all DECnet implementations. DECnet/E queues incoming Interrupt Messages at the head of the pending message queue, behind the first message in the queue (since the user program could be in the process of retrieving the first message) and behind any other pending Interrupt Messages queued for the program.

In the DECnet/E implementation, these Interrupt Messages are not true interrupts in the sense of causing an immediate jump to an interrupt processing routine. (Note that some DECnet implementations do treat Interrupt Messages as true interrupts.) They are just messages that can be identified as different from ordinary data messages and that are placed at the head of the pending message queue (see Figure 2-6).

If a program is designed to send Interrupt Messages, the receiver at the other end of the link must be set up to process them. Interrupt Messages can contain a small amount of user data so that programs can be designed to recognize Interrupt Messages for different purposes.

See Chapters 3 and 4 for a detailed discussion of Interrupt Messages.

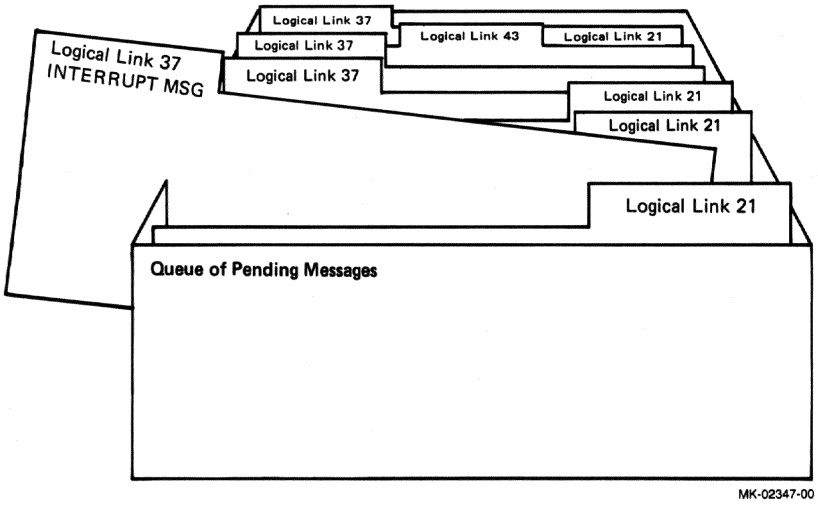


Figure 2-6: Queueing of Interrupt Messages for Logical Links

Flow Control

A message uses system buffer space until it is received by the user program. Thus, pending messages tie up valuable system resources. Several flow control mechanisms are used in DECnet to protect systems from being flooded with messages. Some of these mechanisms are system-regulated while one mechanism (Data Message flow control) is selected and controlled by the user program.

Each receiver program can select one of the following three types of program-regulated flow control:

- Segment flow control
- Message flow control
- No active flow control

See the section "Program-Regulated Flow Control" in Chapter 4.

DECnet/E Logical Link Limitations

The implementation of DECnet for RSTS/E allows a maximum of 127 active logical links for the entire node at any one time. The system manager can reduce this maximum to some number between 1 and 127. There is no limit to the number of logical links per program other than the system maximum.

Chapter 3

Overview of DECnet/E Message Services

RSTS/E provides message services for communication between local programs. The DECnet/E message services are implemented as an extension of these local services. In addition, DECnet/E offers new capabilities for local communication not found with the original RSTS/E message mechanism. The original local communication services are described in the *RSTS/E Programming Manual*. They are repeated in this document both for completeness and because they are the logical basis for the DECnet extensions.

It is the way in which the user interfaces with DECnet from RSTS/E that is implemented here as an extension of local capabilities. An RSX-11M-Plus programmer might interface with DECnet somewhat differently. The underlying DECnet architecture provides the consistency. The programmer at each system uses a familiar language and form. DECnet handles both the transmission itself and the translation necessary at both nodes to convert the original call to a form that can be transmitted over the line and back to the format expected at the receiving node.

Table 3-1 lists the calls used for interprogram communication by programs running under DECnet/E. The remaining sections of this chapter give a general description of each of the calls. Specific high-level language calls, with detailed formats, are given in Part II of this manual, "Programming Information."

Table 3-1: Summary of Calls for Interprogram Communication

Call	Function
Declare Receiver	Registers the program with the operating system for message services.
Remove Receiver	Terminates message operations for the program.
Get Local Node Parameters	Obtains local node information.
Log User Event	Queues a user-generated event to the local event processor for logging.
Send Local Data Message	Transmits user data to a local program.
Send Local Data Message with Privilege Mask	Transmits user data, plus a privilege mask supplied by the monitor, to a local program.
Send Connect Initiate Message	Requests a logical link connection with another program.
Send Connect Confirm Message	Accepts a logical link connection requested by another program.
Send Connect Reject Message	Rejects a logical link connection requested by another program.
Send Network Data Message	Transmits user data to a network program over an established logical link.
Send Interrupt Message	Transmits interrupt data to a network program over an established logical link.
Send Link Service Message	Requests data over a flow controlled logical link.
Send Disconnect Message	Disconnects an established logical link after all pending messages have been sent.

Table 3-1: Summary of Calls for Interprogram Communication (Cont.)

Call	Function
Send Link Abort Message	Disconnects an established logical link immediately, destroying any messages waiting to be sent.
Receive	Receives a message from the queue of pending messages. (One of nine message types, corresponding to the nine Send calls listed above, could be received).

General Remarks

The Declare Receiver call informs the local RSTS/E monitor that the program issuing the call intends to communicate with other programs. This call identifies the program and defines which of the services is required. The program issues a Remove Receiver call when message operations are complete to release any system resources used for message services. The Get Local Node Parameters call returns information to the calling program concerning the local node. The Log User Event call permits a user-written program to queue an event to the event processor for logging.

Beyond these, the operations available fall into two categories: send and receive. The entities sent and received are called messages. They consist of control information identifying the type of message, and in most cases, data of interest to the programs exchanging information. Ten message types are available for these operations (see Table 3-1).

A separate call is used to send each of the message types. That is, there is a Send Connect Initiate Message call, a Send Local Data Message call, and so forth. When a program executes one of these calls, Session Control uses the parameters and user data included in the call to format the appropriate network message into system buffer space. Session Control gives the message to ECL, which puts control information in the message. This message is then queued to the Routing layer for transmission across the network. The way in which a message is queued for transmission depends on the type of message. See the section "Transmit Queue Management" in Chapter 4 for more information on transmission queuing.

The Receive call retrieves a message from the program's queue of pending messages that Session Control maintains in system buffer space. Session Control queues incoming messages for the program as they come from ECL.

Note

Within the current context, local programs are programs running at the local node that send messages using either the Send Local Data Message or the Send Local Data Message with Privilege Mask calls. Network programs, on the other hand, are programs running either locally or remotely that use DECnet to establish and send data over logical links.

The Receive call returns control information identifying the sender and the type of message (see Table 3-1). User data accompanying the message is returned to a user buffer specified in the Receive call where it can then be examined and processed.

A user program can issue general or selective Receive calls. On a general Receive, the first message in the queue is returned. A selective Receive can select the first message from any local sender, a particular local sender, any network sender, or a particular logical link.

There can be times when no pending messages are in the queue when a Receive call is issued. One option available in the Receive call causes the program to sleep, or suspend execution, until some action takes place (for example, until a message is queued or a specified amount of time has passed). You can also issue a Receive call to retrieve only part of a message. A program with limited buffer space can choose to retrieve only part of the user data with one Receive call. Subsequent Receive calls retrieve the rest of the data, or the remaining data can be discarded.

Registering with the Monitor: Declare Receiver Call

A program declares its intent to use the message services by issuing a Declare Receiver call. You must execute this call before a program can send network messages or receive messages of any type. However, a program can send local data messages without executing this call.

However, the Declare Receiver call does more than just declare intent. It also defines the way in which other programs throughout the network are to address this program (the program's identity) and the way in which the program can thereafter use the message services.

The Declare Receiver call has some rather far-reaching effects. For this reason, the parameters specified with the call and how the system uses them are described here in detail.

Declaring Identity

A Declare Receiver call identifies the calling program with a network object type code, a name, or both. The manner in which a program declares its identity affects the way incoming link connections are handled.

A name is an alphanumeric identifier that can be used by either local or network programs to refer to the declarer. No two programs at the local node can use the same name at the same time. If a name is given, it must be unique within the local node.

A network object type code, on the other hand, is a number in the range from 1 to 255. It can be used by other network programs to refer to the declarer. These codes simplify the identification of programs performing identical functions at different nodes of a network, without going through the cumbersome process of arriving at and enforcing standard naming conventions.

Note

> Local programs cannot address a receiver by a network object type code.

A network object type code does not have to be unique. Also, more than one receiver can declare identity with the same network object type code at the same time. A program performing a general-purpose function of interest to the entire network can incur heavy use. By permitting more than one program to declare identity with the same network object type code at the same time, DECnet/E permits execution of multiple copies of a program to meet occasional heavy usage.

A range of network object type codes (1 through 127) is reserved for DIGITAL-supplied network applications. The remainder (128 through 255) are available for user applications programs. See Appendix A for a list of currently assigned network object type codes for DECnet.

Note

Whenever possible in the DECnet environment, DIGITAL recommends that programs declare their identity with and be addressed by a nonzero network object type codes.

See Chapter 4 for a detailed discussion of network addressing.

Declaring Intended Usage

In addition to identifying the program, the Declare Receiver call defines how the program intends to use the message services. For example, a program can limit the incoming messages to those from local senders only. Or, it can limit incoming messages to those from network senders only.

Other protective limits can be set in the Declare Receiver call:

- The maximum number of messages that can be stored in the pending message queue.
- The maximum number of active inbound logical links that the program can have at any given time.
- The maximum number of active outbound logical links that the program can have at any given time.

These limits affect incoming messages and both incoming and outgoing requests for logical links. The Declare Receiver call does not limit the type or number of send calls a program can execute.

DECnet/E Use of the Declare Receiver Call

When a Declare Receiver call is issued, the system sets up a 32-byte Receiver ID Block (RIB) to associate the information with the RSTS/E job number. The RIB actually applies to the job rather than the program that issues the Declare Receiver call. This distinction is rather fine for most applications, but does make it possible, for example, to chain together a set of programs using message services. The first program in the series would execute a Declare Receiver call. Then other programs in the chain would also be valid receivers as long as they were always executed under the same job number.

The RIB holds the arguments passed with the Declare Receiver call, along with other system and Session Control maintained information, such as the address pointers of the pending message queue and the address of the list of active logical links.

Remove Receiver Call

A Declare Receiver call remains in effect until a program issues a Remove Receiver call. The Remove Receiver call releases the RIB. All pending messages are discarded and all active logical links are destroyed. Session Control requests ECL to send a Link Abort Message to the remote programs for any active logical links.

Like the Declare Receiver call, the Remove Receiver call applies to the RSTS/E job. Issue a Remove Receiver call as soon as message services are no longer needed. This practice prevents queuing of unwanted messages and releases the system buffer space used for the RIB.

Also, any program that could be stopped by a terminal user's CTRL/C should have provisions to issue a Remove Receiver call and terminate gracefully. If a job is killed, RSTS/E automatically removes any receiver declarations that the job has.

Note

It is good practice for a program to issue a Remove Receiver call immediately before issuing a Declare Receiver call. This prevents errors on the Declare Receiver call caused by abnormal termination of a previous program running under the same job number.

Get Local Node Parameters Call

The Get Local Node Parameters call returns information to the calling program concerning the local node. This information consists of the node name or number, address, and the node's default project-programmer number (if any).

A general-purpose network program -- one that runs on several different nodes rather than being specific to any one node -- would use this call to obtain the node name or address if it wanted to keep track of that information for any reason (for example, logging). The default account is returned for compatibility with DECnet/E V2.0 which did not automatically log in network jobs.

Log User Event Call

The Log User Event system call permits a user-written program to queue an event to the system event processor for logging. Before an event is logged, it is time-stamped by the system in the standard Network Information and Control Exchange (NICE) protocol.

Optional parameter data can be passed to the event processor for processing and output. If it is included, this data must be in NICE protocol format. See the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for further information on event logging.

Note

This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture. It is provided as a convenience for the experienced network user and is not intended for normal network programs.

Send/Receive Local Data Message

The Send Local Data Message call transfers up to 512 bytes of user message data and up to 20 bytes of parameter data to a local program. The message is placed in the appropriate receive queue of the target program. The target program can be identified either by job number, local object type, or by the name given in its Declare Receiver call.

A Receive call that returns a Local Data Message returns control information identifying it as a Local Data Message from a specific local sender, and up to 512 bytes of user data and 20 bytes of parameters.

The Local Data Message is not used for DECnet communication but is included here for reference. A user program can carry on local conversations and maintain one or more DECnet logical links at the same time.

Send Local Data with Privilege Mask

A local job can send its privileges to a declared receiver in the same way as the Send Local Data Message call. The Send Local Data Message with Privilege Mask call sends a data message plus a privilege mask

supplied by the RSTS/E monitor. This call provides a method for a program to tell another program about a job's current privileges and guarantees that the data cannot be falsified.

Send/Receive Connect Initiate Message

The Send Connect Initiate Message requests a logical link connection with a remote program.

The target node and program are identified by a Connect Data Block passed as part of the call. The following information is specified with the call:

- The User Link Address (ULA) that the source program uses to refer to this link.
- The type of receiver flow control desired by this program -- that is, the type of flow control to be imposed on incoming data messages. See Chapter 4 for a full discussion of flow control.
- A receive maximum, which is the maximum amount of data to be received by the program over this link in one Network Data Message. If the maximum buffer size set by Network Management is not large enough to handle the receive maximum as specified by the user program, Session Control alters the receive maximum to the smaller limit before forwarding the Connect Initiate Message. The local program is informed of this change (if any is made) by a field in the Connect Confirm Message from the remote program when it accepts the link.
- A Connect Data Block identifying the target node and program to which the connection is requested. The target program can be identified by name or network object type code. Session Control inserts the local program identification, as specified in the source program's Declare Receiver call, into the Connect Data Block. This allows the remote program to determine who sent the connect request and decide whether to accept or reject the connection. The Connect Data Block can also contain data required by the remote program or system for access control. Such data might consist of a project-programmer number, password, and account number -- or whatever is required by the remote program or system.
- Up to 16 bytes of user data.

A Receive call that returns a Connect Initiate Message from the queue of pending messages returns control information identifying it as a Connect Initiate Message from a specific network program, and requirements imposed by the remote program. The information includes the following:

- The Local Link Address (LLA) assigned by the local ECL to refer to this logical link. The program uses the LLA to refer to the link in its responding Connect Confirm or Connect Reject Message.

  Note that the LLA is only used to refer to a link while no ULA is in effect. The ULA is established by the receiving program in its Connect Confirm Message that accepts the connect request and is used thereafter by the program to refer to the link.

- The type of receive flow control desired by the remote program -- that is, the type of flow control that is imposed on outgoing data messages over this link.

- A receive maximum, which is the maximum amount of user data that the local program can receive over the link. This maximum is calculated by the local ECL, and based on the size of receive buffers it allocates. The program can reduce this maximum further in its responding Connect Confirm Message.

- A transmit maximum, which is the maximum amount of user data that the program can transmit over this logical link in one Network Data Message. This maximum is imposed by either the remote ECL, Session Control, or the remote program and corresponds to the remote program's receive maximum.

- A Connect Data Block, identifying the remote node and program requesting the link. The Connect Data Block also indicates how the remote program addressed this one (by name or network object type code). For compatibility with DECnet/E V2.0, network objects declared with NCP as VERIFICATION OFF are passed access control information from the remote program, such information would also be included as part of the Connect Data Block. This feature should not be used for new program development, as it is not as secure as having inbound access checking done by Session Control. The Connect Data Block and up to 16 bytes of user data are delivered to the user buffer specified in the Receive call.

Send/Receive Connect Confirm Message

The Send Connect Confirm Message call accepts a logical link requested by a remote program. That is, it is a positive response to a received Connect Initiate Message. The following information is specified with the call:

- The ULA by which the program now refers to this link.
- The LLA that identifies the link being accepted. Note that the LLA is taken from the Connect Initiate Message received for this particular link.
- The type of receive flow control desired by the calling program for incoming data messages over this link. See Chapter 4 for a full discussion of the flow control options.
- A receive maximum, which is the maximum amount of user data that the program wants to receive as a unit (that is, as one Network Data Message).
- Up to 16 bytes of user data.

A Receive call that returns a Connect Confirm Message from the pending message queue returns control information identifying it as such, along with requirements for messages to be transmitted over the link. The information includes the following:

- The ULA established by the receiving program in the Connect Initiate Message that requested the link. This identifies the particular link being confirmed.
- The type of receive flow control requested by the remote program -- that is, the type of flow control imposed on outgoing data messages. See Chapter 4 for a full discussion of the flow control options.
- A receive maximum, which is the maximum amount of user data that the local program receives on this link in one Network Data Message. This is the same or less than the size requested in the program's Connect Initiate Message. A smaller size is indicated by the local ECL if it cannot handle the size requested in the Connect Initiate Message.
- A transmit maximum, which is the maximum amount of data that the local program can send to the remote program as a unit (with one Network Data Message) over this link.
- Up to 16 bytes of user data, delivered to the user buffer specified in the Receive call.

Send/Receive Connect Reject Message

A Send Connect Reject Message call rejects a logical link requested by a remote program. That is, it is a negative response to a received Connect Initiate Message. The following information is specified with the call:

- The LLA assigned by the local ECL to identify the link being rejected. The LLA is taken from the received Connect Initiate Message.
- A code identifying the reason for the rejection. Ordinarily, this code is zero unless the program is rejecting the connection due to some standard reason, such as "invalid accounting data." See Appendix B for a list of Session Control rejection codes.
- Up to 16 bytes of user data.

A Receive call that returns a Connect Reject Message from the queue returns control information identifying it as such and the following information:

- A ULA identifying the link being rejected. This ULA is the same as that established by the local program in the Connect Initiate Message that requested the link. Either the remote Session Control or the remote program could have rejected the link.
- A code identifying the reason for the rejection if the link was rejected by the remote Session Control, ECL, or by the remote program for some standard reason, such as "invalid accounting data." See Appendix B for a list of Session Control rejection codes.
- Up to 16 bytes of user data, if the link was rejected by the remote program for some nonstandard reason. This data is delivered to the user buffer specified in the Receive call.

Send/Receive Network Data Message

The Send Network Data Message call transmits user information to a remote program over an established logical link. The logical link is identified by the ULA established by the calling program in its Connect Initiate or Connect Confirm Message. The amount of user data that can be sent is limited to the transmit maximum established by the remote program (see the previous discussion of a received Connect

Initiate or Connect Confirm Message). Sending a Network Data Message is subject to flow control imposed by the system and by the remote program, as indicated by the flow control option selected by the remote program. A program can request that link status information be returned by a Send Network Data call and use the information returned to schedule subsequent send and receive requests. See Chapter 4 for a full discussion of flow control.

A Receive call that returns a Network Data Message returns identifying control information and delivers the user data to a buffer specified in the call. The amount of user data received in one Network Data Message does not exceed the receive maximum established by this program for the logical link (see the previous discussion of a sent Connect Initiate or Connect Confirm Message). Incoming Network Data Messages are subject to system flow control and to the flow control option, if any, requested by the local program in its Connect Initiate or Connect Confirm Message. If the program requests flow control, it must send Link Service Messages to request data from the remote program (see Chapter 4).

Send/Receive Interrupt Message

The Send Interrupt Message call transmits high-priority data to a remote program over an established logical link. All DECnet systems maintain separate data streams for Interrupt and Data Messages. The Interrupt Message is delivered to the remote program ahead of all Data Messages waiting to be processed at the remote end. You can send up to 16 bytes of user data in an Interrupt Message. Sending an Interrupt Message is subject to flow control imposed by the remote system or program. The local program can request that link status information be returned following a Send Interrupt Message call and use the information returned to schedule subsequent send and receive requests. See Chapter 4 for a full discussion of flow control.

A Receive call that returns an Interrupt Message returns identifying control information and delivers up to 16 bytes of user data to a buffer specified in the call. The Interrupt Message is queued at the head of the pending message queue, behind the first Data Message and behind any other pending Interrupt Messages queued for the program. Incoming Interrupt Messages are subject to flow control imposed by the local ECL. After receiving each Interrupt Message, the local program must send a Link Service Message to reenable incoming Interrupt Messages from the remote program over a logical link (see Chapter 4).

Send/Receive Link Service Message

The Send Link Service Message call requests data over a logical link -- either to request data from a remote program over a flow controlled link, or to reenable incoming Interrupt Messages from the remote program. In these two cases, detailed link status information can be returned or the call can be issued solely to obtain this information.

When a link becomes unblocked, a Receive call returns a Link Service Message. If a local program tries to send a Network Data or Interrupt Message and fails because the remote system or program has inhibited flow, the local ECL requests Session Control to inform the local program when the condition clears. To encourage efficient processing of pending messages, a Link Service Message is only returned on a Receive call when the pending message queue is empty. Session Control also returns detailed link status information that the local program can use in regulating its send and receive requests. Note that the format of the status information for a received Link Service Message is the same as that returned, upon request, for a Send Network Data, Interrupt, or Link Service call. See the section "Format of Received Link Service Message" in Chapter 5 (BASIC-PLUS), Chapter 6 (BASIC-PLUS-2), Chapter 9 (extended COBOL), Chapter 10 (FORTRAN-IV), or Chapter 11 (MACRO).

See the discussion of flow control in Chapter 4 for a more complete understanding of the uses of Link Service Messages.

Send/Receive Disconnect Message

The Send Disconnect Message call breaks an established logical link provided that all Network Data, Interrupt, and Link Service Messages previously sent over the link have been acknowledged by the remote ECL. If these messages have not been acknowledged, the call terminates with an error and the link is not broken. Successful completion of the call means that no more messages can be sent over the link. The ULA is freed and can be used to establish another logical link. However, messages already in the pending message queue remain. They can (and should) be cleared from the queue by doing selective Receive calls using the user link address of the disconnected link. Once the link has been disconnected, ECL discards any new messages received for the link. The Disconnect Message is sent to the remote system. When it is received by the remote ECL, the link is broken at that end. Up to 16 bytes of user data can be sent with a Disconnect Message.

A Receive call that returns a Disconnect Message from the pending message queue returns control information identifying it as such, as well as a user link address and up to 16 bytes of user data. Receipt of a Disconnect Message indicates that the remote program has

terminated the logical link. Furthermore, it indicates that the local Session Control placed all previous Network Data Messages, Interrupt Messages, and Link Service Messages sent by the remote program in the pending message queue before the Disconnect Message was received. Once the local ECL receives the Disconnect Message, the local program cannot send any more messages over the link.

The Disconnect Message is useful in master-slave communication when the master program only transmits data and the slave program only receives data. Since there is no two-way communication, the master program can issue a Disconnect Message to break the link with the assurance that the remote system has received all the data sent and queued it for the slave program. However, successfully completing the disconnect does not guarantee that the slave has processed the data. Where such guarantees are desirable, or where two-way communication is desired, programs should agree to terminate the logical link by exchanging Network Data Messages, Interrupt Messages, or whatever.

Send/Receive Link Abort Message

A Send Link Abort Message breaks a logical link immediately. Unlike a Disconnect Message, ECL makes no attempt to make sure that the remote system has acknowledged messages previously sent over the link. In fact, ECL discards any messages it has queued for transmission for the link (see the section "Transmit Queue Management" in Chapter 4). No further data can be sent over this logical link. Received messages already in the pending message queue remain; however, new messages received after the Link Abort Message is issued are not queued for the link. Messages already in the queue can (and should) be cleared by doing selective Receive calls using the user link address of the aborted link.

If the link is established, the Link Abort Message is sent to the remote system as notification that the link is broken and up to 16 bytes of user data can accompany the Link Abort Message. A Link Abort Message can also be sent for a logical link that has not yet been confirmed by the remote program. In this case, ECL breaks the link at the local node, but it does not notify the remote system. Thus, the user data does not reach the remote program. If the remote program confirms the link, the local ECL no longer knows the link and it so informs the remote ECL, which instructs Session Control to notify the remote program. Should the remote program reject the link, the local ECL informs the remote ECL that it has no such link but since the remote program rejected the link anyway, no notification is given to the remote program.

A Receive call that returns a Link Abort Message from the pending message queue returns control information identifying it as such, as well as a ULA and a LLA to identify the link aborted. The LLA

identifies a link aborted during the initial connection sequence. For example, if the local ECL received a Connect Initiate Message, but the local program has not yet responded with a Connect Confirm or Connect Reject Message, a Link Abort Message is queued with the same LLA as was specified in the received Connect Initiate Message. The link could have been aborted by the remote ECL, Session Control, or the remote program. If the remote ECL or Session Control aborts the link, the message contains a reason code identifying the reason for the rejection. If the remote program aborts the link, up to 16 bytes of user data can also be delivered with the Link Abort Message. Once the local ECL receives the Link Abort Message, the local program cannot send any more messages over the link.

Chapter 4

Network Addressing and Flow Control

With the background information from Chapters 2 and 3, this chapter can begin to discuss two areas of interprogram communication in a DECnet network:

- **Network addressing** -- Concerns the methods whereby a program declares its identity for the purpose of receiving network messages.
- **Flow control** -- Involves the various techniques used by DECnet to ensure an orderly transfer of message and control data between nodes in a network.

Network Addressing

When a program declares its intent to use the network message services, the following DECnet/E features influence it:

- The options available in the Declare Receiver call.
- The way in which DECnet/E Session Control handles incoming requests for logical link connections.
- The multiple copy execution feature, by which more than one copy of a network program can be executed in the RSTS/E timesharing environment.
- The automatic job start-up feature, by which DECnet/E allows programs to be defined by the system manager for automatic start-up on receiving a request for a logical link connection.

Declaring Network Names and Object Type Codes

Every DECnet/E program that wants to receive messages from local senders or establish logical links for data exchange with other network programs must declare its identity in a Declare Receiver call. The program can declare its identity with:

- A nonzero object type and with a null name -- Abbreviated to DR(n,null) in this chapter
- A zero object type and a nonnull name -- DR(0,name)
- A nonzero object type code and a nonnull name -- DR(n,name)

Remember from the discussion in Chapter 3 that you should use nonzero object type codes consistently throughout the network.

Declaring identity by object type code alone -- DR(n,null) -- permits multiple copies of a program to be executed without the necessity of generating a unique identity for each copy. DECnet/E allows more than one program to run and declare identity with the same object type code.

On the other hand, declaring identity by name alone -- DR(0,name) -- requires uniqueness in the sense that no other program at the local node can declare identity with the same name at the same time. If a program declares its identity with the same name each time it runs, multiple copies cannot execute simultaneously.

However, it is possible to combine the features of uniqueness and of multiple copy execution. By appending the RSTS/E job number to a common root, a program can create a unique, nonnull name (for example, FAL021, FAL005). The program can then declare its identity with both the name created and a nonzero object type code. This permits multiple copies of the program to be run as a result of incoming connect requests but permits each copy to create its own unique identity.

Handling of Incoming Connect Requests

When a Connect Initiate Message is received at the local node, Session Control searches its list of RIBs for the proper receiver. All other incoming messages reference an established logical link and are queued to the correct program without searching the list of RIBs. The method of addressing specified in the Connect Initiate Message determines how Session Control processes the request.

Incoming requests for logical link connections can be addressed to either an object type code alone -- CI(n,null) -- or to a name alone -- CI(0,name). DECnet/E does not allow incoming connect requests to be addressed using both object type code and name. It rejects such requests, sending back a Connect Reject Message with a reason code indicating that the requested program cannot be found.

Incoming Connect Request Addressed to Object Type Code -- CI(n,null)

If the incoming Connect Initiate Message is addressed to an object type code -- CI(n,null) -- Session Control checks its RIBs to see if one or more programs are running that have declared identity with this object type code. The Declare Receiver call could have specified either DR(n,null) or DR(n,name).

If Session Control finds one or more such receivers that have not reached their declared logical link maximum or message maximum, it queues the connect request for the first such receiver in its list.

Note

A program sets its own limits on the number of logical links allowed and the size of the pending message queue in the Declare Receiver call (see Chapter 3).

If there is no such receiver or if there are one or more such receivers but all have reached their declared logical link maximum or message maximum, Session Control attempts to start another copy of the program (see "Automatic Job Start-Up"). Briefly, Session Control checks to see if the program identified in the Connect Initiate Message has been defined by the system manager for automatic start-up. If so, Session Control creates a job and starts the program. The program must then issue a Declare Receiver call with the appropriate object type code. This creates a RIB and a pending message queue to which the waiting Connect Initiate Message can then be transferred. If the program does not issue a Declare Receiver call within two minutes, Session Control rejects the link by returning a Connect Reject Message to the remote program.

Incoming Connect Request Addressed to Name -- CI(0,name)

When the incoming Connect Initiate Message is addressed to a name -- CI(0,name) -- ECL checks its list of RIBs to see if a program is running that declared its identity with this name. The Declare Receiver call could have specified either DR(0,name) or DR(n,name).

If such a receiver program is running, and it has not yet reached its declared logical link maximum or message maximum, ECL places the Connect Initiate Message in the receiver's queue of pending messages.

If no such program is running, ECL attempts to start the program. If it is successful in doing so, the program started must declare its identity with the appropriate name. Again, if the program fails to issue a Declare Receiver call within two minutes, Session Control rejects the connection.

If such a program is running, but it has reached its declared logical link maximum or message maximum, Session Control rejects the connection, returning a Connect Reject Message with an appropriate reason code explaining the rejection. Session Control does not attempt an automatic start-up in this case, because a program is already running with the name to which the Connect Initiate Message was addressed. Since receiver names must be unique, ECL cannot generate a second copy with the same name.

Multiple Copies

In a DECnet/E system, multiple copies of a program can be started in the following ways:

- By a local terminal user
- Through the Print/Batch Services (PBS) package
- Through the automatic job start-up feature

The manner in which copies of a program can be started has some bearing on the way in which a Declare Receiver call should be issued.

Multiple Copies Initiated by Terminal Users

A program run from a local terminal with the purpose of communicating with a remote program normally sends at least the initial Connect Initiate Message requesting a logical link. It is unlikely that remote users would want to have the success of incoming link connections rely upon someone starting the program manually.

In this case, the program can declare its identity with DR(n,null), or with DR(n,name) or DR(0,name) as long as the program generates a unique name with each copy (see "Automatic Job Start-Up"). The choice depends on whether the program receives any connect requests, and if so, if it is necessary for the remote program to connect with a specific copy.

For example, suppose that the local program issues the initial Connect Initiate Message for establishing the first logical link. Suppose further that it is then necessary for the remote program to establish a second logical link with the same copy that sent the Connect Initiate Message for the first link. The local program can declare its identity with either the form DR(0,name) or DR(n,name). The unique copy name is passed to the remote program in the Connect Data Block in the Connect Initiate Message. The remote program can then address its connect request in the form CI(0,name) to make sure that the request is delivered to the same copy that issued the first Connect Initiate Message.

Multiple Copies Executed under PBS

Multiple copies of a program can also be executed through use of PBS (see the *RSTS/E System User's Guide*). Again, such programs are normally responsible for initiating at least the first logical link connection. The techniques described in the previous section for multiple copies started by terminal users would also apply here.

Multiple Copies Initiated through Automatic Start-Up

Multiple copies of a program can also be initiated for programs that the system manager has designated for automatic job start-up. Such copies result from incoming connection requests addressed CI(n,null) -- see the section "Automatic Job Start-up" for full details. The local program can declare its identity with DR(n,null), or with DR(n,name) as long as the name is unique for each copy.

Automatic Job Start-Up

Using the Network Control Program (NCP), the system manager can install user programs for automatic start-up. Such programs reside as files on disk in the normal fashion and can be started by Session Control to fulfill incoming requests for logical link connections (see "Handling of Incoming Connect Requests"). See the *DECnet/E System Manager's Guide* for a complete description of this installation method.

Briefly, a DEFINE OBJECT or SET OBJECT command in NCP relates the file name to an object type code, a name, or both. These commands also allow two parameters to be associated with a program. The second parameter is made available to the program when it is started automatically. BASIC-PLUS or BASIC-PLUS-2 programs use the first parameter as a starting line number. MACRO programs use it as an entry parameter. COBOL and FORTRAN-IV programs ignore it.

Session Control maintains a list of these definitions in its parameter file. The definitions are ordered by object type code, and thus, there can be only one program defined for automatic start-up for each object type code. In particular, there can be only one program defined for automatic start-up with an object type code of zero -- (0,name).

When a program must be started automatically, Session Control checks the object type code or name given in the incoming connect request against its list of defined programs. If a match is found, Session Control executes a monitor request to run the program as a detached job under RSTS/E.

Once started, the program must declare its identity with the object type code or name that was specified in the Connect Initiate Message that originally caused the program to be started. If it does not, the connect request is never delivered to the proper pending message queue.

Session Control passes this information to the started program in core common. Session Control also passes any optional parameter established by the DEFINE OBJECT or SET OBJECT command. BASIC-PLUS and BASIC-PLUS-2 users can access core common directly with the Get Core Common system call. MACRO users can access core common directly. FORTRAN and COBOL users cannot access core common directly, but can use a MACRO-11 subroutine to retrieve the data (see the *RSTS/E System Directives Manual*). The format of the core common string is:

Byte(s)	Contents
CORCMN+0	Length of data in core common
+1	Object type code (0-255)
+2-7	Name (ASCII)
+8-9	Second DEFINE parameter (-32768 through 32767)
+10	Reserved

The mnemonic CORCMN is assigned by the file COMMON.MAC to octal location 460 (see the *RSTS/E System Directives Manual*).

If the object type code in byte 1 of core common is nonzero, the program must declare its identity with that object type code. It can also use a nonnull unique name, if desired. If the object type code is zero, the program must declare its identity with the name given in

bytes 2-7 of core common. It can also declare its identity with a nonzero object type, if desired.

When a program issues a Declare Receiver call, Session Control transfers the waiting connect request to the program's queue of pending messages as a Connect Initiate Message. Session Control waits up to the parameter INCOMING TIMER from the time it starts the program for the connect request to be transferred.

If the connect request is not transferred to a queue within this time period, Session Control rejects the connection, sending a Connect Reject Message to the remote program. Since the remote program could also abort the link within this period, the first action of an automatically started program (after issuing the Declare Receiver call) is to issue a Receive call to check for pending messages. If there are no messages, the program issues a Remove Receiver call, and terminates execution with the Logout system call.

Note

> The Logout system call is not directly accessible to FORTRAN and COBOL but can be executed through a MACRO-11 subroutine using the UU.BYE system call (see the *RSTS/E System Directives Manual*).

Once the connect request has been received, the program responds to the request immediately. While the local Session Control's timer is stopped when the message is transferred to the receiver's pending message queue, the remote node could be executing a timeout on the Connect Initiate Message.

Flow Control

DECnet flow control protects user programs as well as entire systems from being flooded with more messages than can be processed at any given time.

A DECnet/E sender is subject to flow control regulations imposed by the local DECnet implementation, the remote DECnet implementation, and the remote program (if the remote program requested flow control when the link was established). As a receiver, a DECnet/E program can select from three flow control options for a logical link to regulate the amount of data sent to it by the remote program.

As a sender and/or receiver, a program must deal with the following four aspects of DECnet/E operation:

- **Transmit queue management** -- Imposed by the local (sending) ECL, makes sure that a local program does not use too much system buffer space for transmission.
- **Backpressure flow control** -- Imposed by the remote system, is used as a safeguard against excessive incoming data messages.
- **Data Message flow control** -- Imposed by receiving user programs, enables a program to control the flow of data to it. Both programs using a logical link can (independently) choose to receive network messages only when they specifically request data from the other program. Or each can choose no flow control, placing no restrictions on the amount of data that they receive.
- **Interrupt Message flow control** -- Imposed by the remote system, protects against Interrupt Message congestion. A DECnet/E program can receive only one unsolicited Interrupt Message over a link. After that, the program must request an Interrupt Message before the remote program can send one.

Transmit Queue Management

Once DECnet/E establishes a logical link, it maintains two transmit queues for the link: one for Network Data Messages and another for Interrupt and Link Service Messages. Both queues hold messages waiting for acknowledgment from the remote ECL.

For example, when a DECnet/E program sends a Network Data Message to a remote program over a logical link, the message is queued to the Routing software for transmission and is also placed in the data transmit queue. The Network Data Message is held there until the local ECL receives an acknowledgment from the remote ECL indicating that it has received the message.

If ECL does not receive an acknowledgment within a specified timeout period, it assumes that the message has been lost and requeues it to Routing for retransmission. If the message has not been successfully transmitted after a specific number of retries, ECL aborts the link with a reason code indicating that the remote system is not responding.

The timeout period used depends on the message type. For outgoing Connect Initiate Messages, ECL uses a predetermined value. However, once ECL establishes a logical link, network management dynamically

computes the timeout period using various network parameters that the system manager has defined by means of NCP's SET EXECUTOR command (see the *DECnet/E System Manager's Guide*).

This acknowledgment procedure between the two ECLs makes sure that messages sent over the logical link have been received, and that they were received in the same order they were sent.

Each of the transmit queues for a logical link can hold up to eight messages. (This limit can be changed by the system manager to any number from one to eight to further protect system buffer space.) When one of the transmit queues for a logical link is filled with messages waiting for acknowledgment, further transmission of that type of message over the logical link is inhibited by the local ECL until one or more of the queued messages have been acknowledged by the remote ECL.

If a program issues a send request while the transmit queue for that message type is full, ECL returns an error to the program. This condition is temporary. When the condition clears, send requests are again allowed for that message type. A program getting such an error on a send request should wait 2 to 5 seconds and retry the send request.

The state of the transmit queues also affects the sending of Disconnect Messages. If any messages are waiting for acknowledgment in the Data or Interrupt/Link Service transmit queue, a Send Disconnect Message call for that logical link is not allowed and results in an error. Thus, a program sending a successful Disconnect Message can assume that all previously sent messages have been received and acknowledged by the remote ECL. However, a Link Abort Message for that logical link is allowed, and ECL discards any messages waiting in the transmit queues.

Figure 4-1 shows transmit queue flow control. PROGA has established two logical links, and PROGB and PROGC have each established one logical link. Each logical link has its own Data and Interrupt/Link Service Message transmit queues. PROGA issues a Send Network Data Message system call. The message goes to the Data Transmit Queue for that logical link and is also queued for transmission by the communication hardware. At the device driver level, all network messages (both data and control messages) await transmission over the physical line.

PROGB is shown trying to send a Network Interrupt Message. However, the Interrupt/Link Service transmit queue for that link is full so ECL returns an error on the send. PROGC issues a Disconnect Message for its logical link. Since messages are still waiting for acknowledgment in the transmit queues, ECL does not allow the disconnection and returns an error. Note that sending a Link Abort Message would have broken the link but the messages would have been discarded.

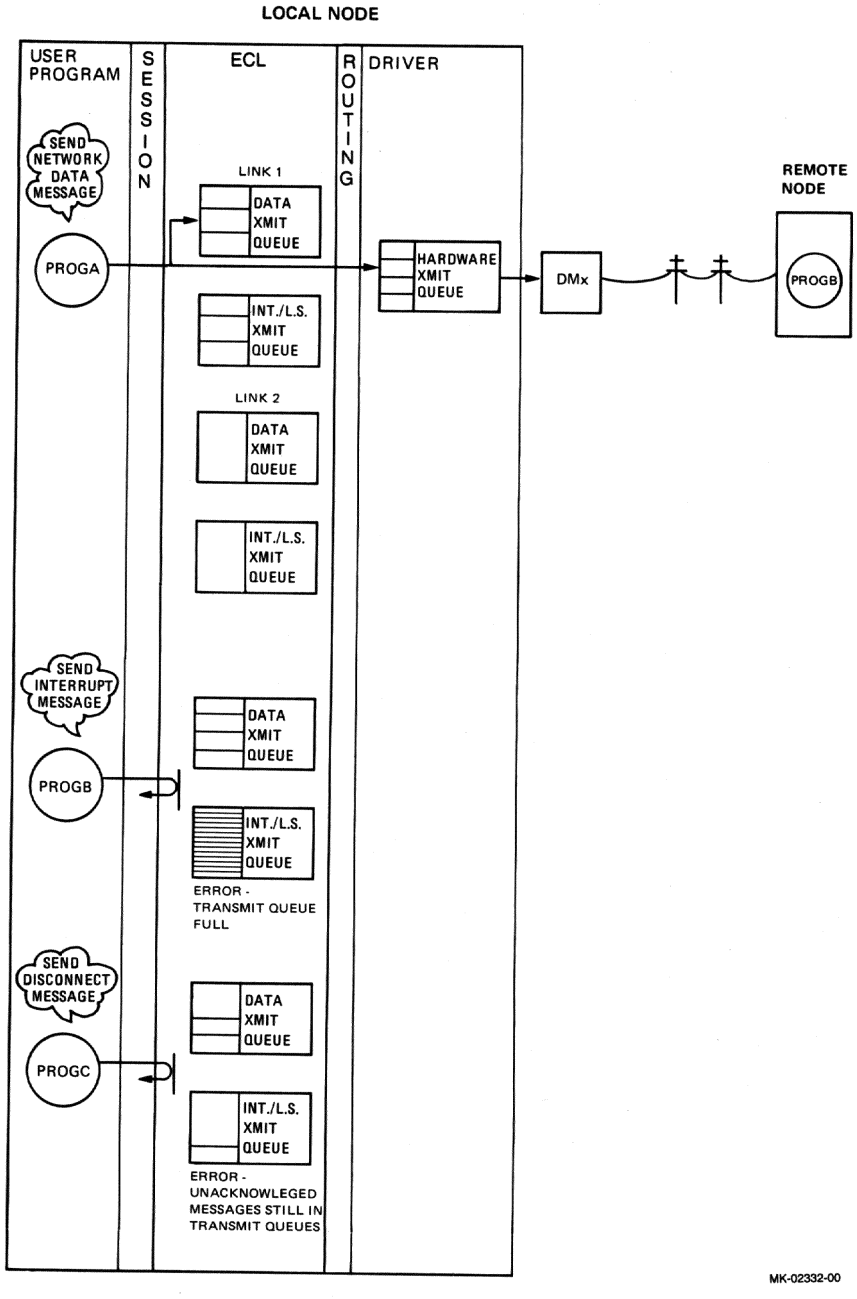


Figure 4-1: How Transmit Queue Flow Control Affects a DECnet/E Program

Backpressure Flow Control

ECL at the remote node can inhibit the local program from sending Network Data Messages over a logical link. The reasons for this blocking can vary from system to system: lack of system buffer space, system load, or some similar condition. This type of flow control, whereby the remote ECL can inhibit message flow from the local program, is known as backpressure.

When the remote ECL receives a Network Data Message that it cannot process, it sends a message to the local ECL that data message flow to the remote program is being inhibited on the link over which the message was sent. The local ECL then inhibits transmission of Network Data Messages over the indicated logical link until further notice is received from the remote ECL. (Note that the local program can still receive messages over the logical link.) When the remote ECL informs the local ECL that the backpressure condition has cleared, the local ECL transmission of Network Data Messages over the logical link.

If the backpressure condition clears before the local program attempts another send over the link, the program is not informed that the condition occurred. However, if the program attempts to send another Network Data Message while transmission on the link is still inhibited, the local ECL returns an error to the program (see A, in Figure 4-2). When the condition clears, the local ECL delivers a Link Service Message to the program, informing it that transmission of Network Data Messages is again allowed for the indicated logical link (see B, in Figure 4-2). This Link Service Message is delivered to the local program on a Receive call when the queue of pending messages is empty.

Note that the remote ECL enforces flow control for specific logical links, not for the entire node.

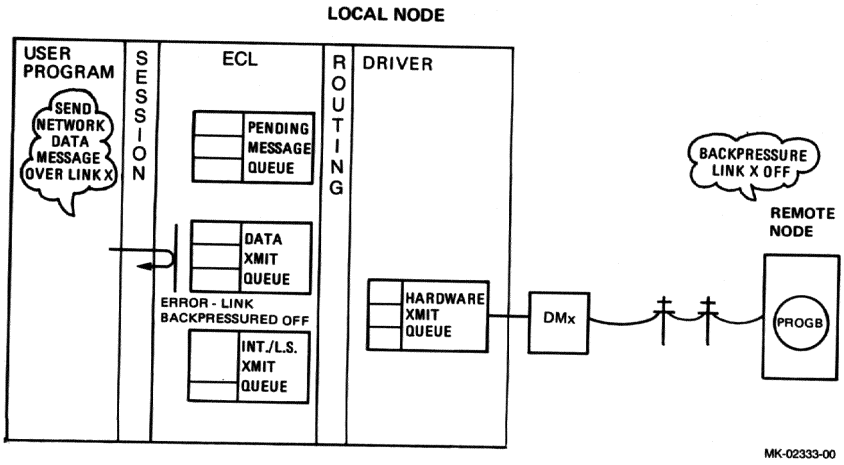
From the reverse viewpoint, the local ECL inhibits incoming data messages on a logical link when a Network Data Message is received from a remote program over the link and the pending message queue of the intended receiver is full. Remember that the local program itself limits the size of its pending message queue in its Declare Receiver call. The local ECL informs the remote ECL that incoming data message flow has been inhibited by sending a message to the remote system.

Furthermore, the local ECL marks all other logical links on the RIB to be turned off if any Network Data Messages are received on those links. However, no message is sent to the remote system unless a Network Data Message is received on one of these links before the restriction is lifted. (Note that, while the local ECL no longer accepts incoming messages, the local program can still transmit.)

When the local program empties its pending message queue, any links that were actually turned off are turned back on and a message is sent

to the remote ECL to reenable transmission. Those links that were merely flagged to be turned off are restored to normal operation, but no message is sent to the remote system.

A. Remote node backpressures a link off



B. Remote node clears backpressured link, local program notified on receive

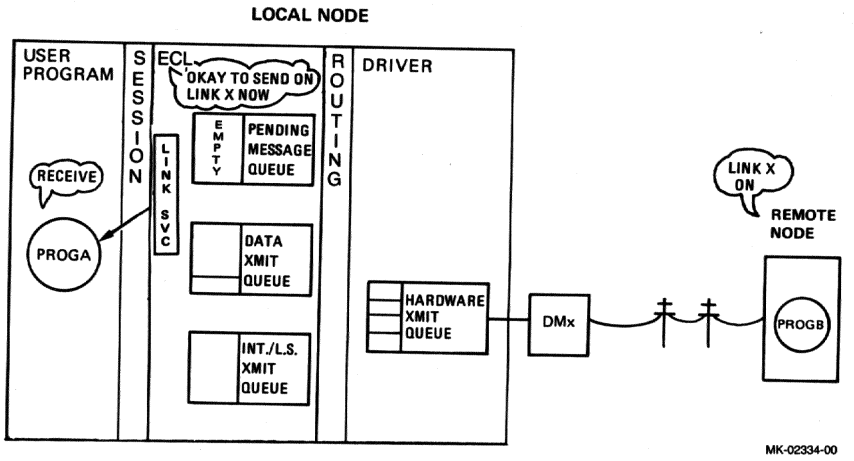


Figure 4-2: How Backpressure Flow Control Affects a DECnet/E Program as a Transmitter

Program-Regulated Flow Control

During the connection sequence that establishes a logical link, each program must select one of the following three flow control options to regulate incoming data over the link:

- **Segment flow control** -- The program requests data from the remote program by indicating how many segments it accepts at

any given time. A segment is defined as the amount of user data transferred in one Network Data Message. The maximum size of a segment is determined by the receive maximum established in the Connect Initiate or Connect Confirm Message. However, the sending program is not required to use this maximum, and can send smaller segments. Note also that segments transmitted and received by the same program are not necessarily equal in size.

- **Logical message flow control** -- The program requests data from the remote program by indicating how many logical messages it accepts at any given time. A logical message can consist of one or more data segments. Control flags, included as part of a Network Data Message, indicate whether the data segment is the beginning, middle, end, or sole segment of a logical message.

- **No active flow control** -- The program sets no limits on the rate at which messages can be sent to it over a logical link.

If the program selects segment flow control or logical message flow control, the remote ECL does not send Network Data Messages over the link until the local program requests such transmission. Requests are made by sending a Link Service Message requesting a specific number of segments or logical messages.

If the program selects the third option (no active flow control), no Link Service Messages can be sent except those requesting Interrupt Messages. Network Data Messages are transmitted by the remote ECL as the remote program sends them, subject to backpressure. DIGITAL recommends the use of no active flow control, to reduce the number of control messages the systems must exchange.

Flow control selection is independent at both ends of the logical link. That is, the remote and local programs need not select the same type of flow control. However, the option selected by the remote program affects how the local program can send Network Data Messages. Therefore, the following discussion of flow control is presented from two viewpoints:

- How the option selected by the local program affects incoming Network Data Messages

- How the option selected by the remote program affects the way the local program sends Network Data Messages

Flow Control on Incoming Data

If the local program selects logical message flow control in the Connect Initiate or Connect Confirm Message establishing a logical link, the local program must request data from the remote program by sending Link Service Messages specifying the number of logical messages to be transmitted.

When a program issues a Send Link Service Message call, the local ECL adds the value specified to a message counter maintained for the link. The local ECL also sends the information on to the remote ECL. The remote ECL likewise adds the value specified to a message counter it maintains for the link. Thus, both ECLs count the outstanding message requests. Each time the remote ECL sends the end segment of a logical message, it decrements the message counter. When the counter reaches zero, the remote ECL inhibits transmission until the local program sends another Link Service Message specifying another value to add to the message counter.

The local ECL uses its message counter as a protective measure. Each time it receives a data segment with the end-of-logical-message flag set, it decrements the message counter. When the counter reaches zero, the local ECL no longer accepts data segments over the link.

When the local program selects segment flow control in the Connect Initiate or Connect Confirm Message, the local and remote ECLs maintain segment counters for the logical link. The remote program can still send segments with the logical message control flags set for the local program to process internally, but the counters maintained by ECLs are kept for data segments sent, not for logical messages. Data is requested by sending a Link Service Message indicating the maximum number of segments to be sent. The local and remote ECLs add this value to their data segment counters and then decrement the counters as the remote program sends segments. When the data segment counter reaches zero, transmission over the logical link is suspended until another Link Service Message increments the count.

When no active flow control is requested in the Connect Initiate or Connect Confirm Message, segments from the remote program can be transmitted over the logical link without being requested by the local program. The local and remote ECLs do not keep counters to regulate data sent and they do not use Link Service calls to request transmission of data. However, Data Message flow can still be restricted by backpressure.

Flow Control on Outgoing Data

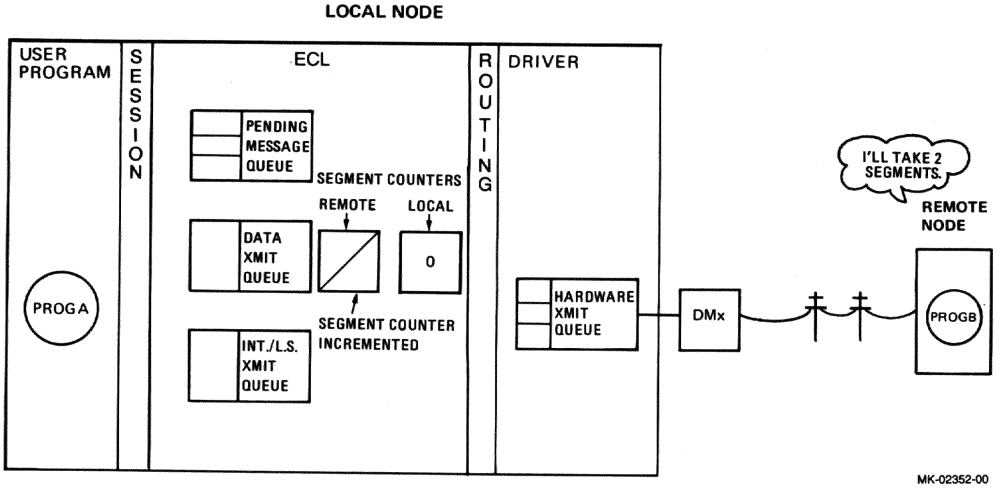
When the remote program specifies a flow control option in its Connect Initiate or Connect Confirm Message establishing the logical link, the

local ECL notes the option and establishes the appropriate counter. The counter is increased when a request for data is received from the remote system or program. If the remote program requested segment flow control, the counter is decremented by one each time the local program issues a Send Network Data Message call. If the remote program requested logical message flow control, the message counter is decremented by one each time a Network Data Message is sent that is flagged as the end data segment of a logical message.

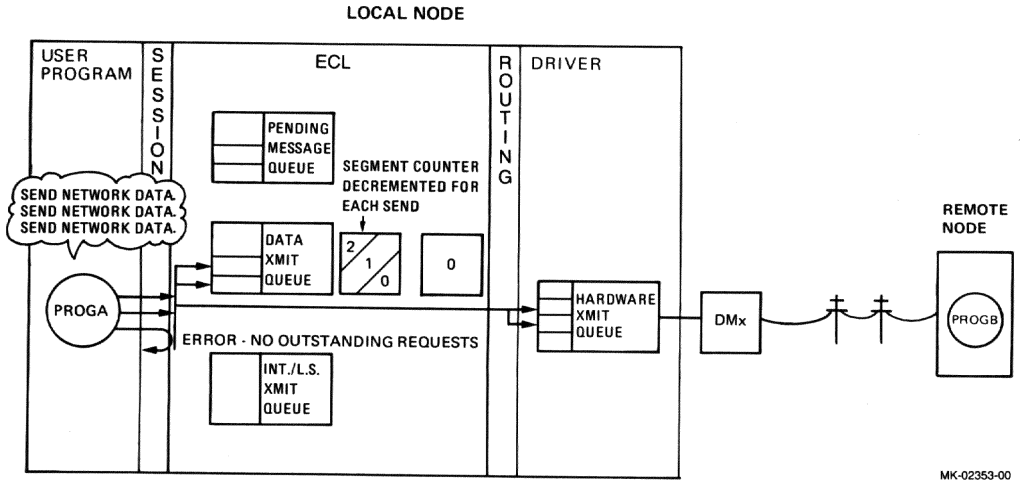
When the counter equals zero, the local ECL inhibits further transmission of Network Data Messages over the logical link, returning an error if the local program tries to do so. In this circumstance, the local ECL delivers a Link Service Message to the local program when the condition clears and the pending message queue is empty. However, a Link Service Message is not delivered if the local program did not attempt to send a Network Data Message while the request count was zero.

Figure 4-3 shows this process. The remote program has requested segment flow control. First, the remote program requests two segments (see A, in Figure 4-3). The segment counter for the remote program is incremented, but since PROGA has not had transmission inhibited, no Link Service Message is queued. The local program issues three Send Network Data Message calls (see B, in Figure 4-3). The segment counter is decremented to one with the first send, and to zero with the second send. The third send, while the counter is zero, results in an error. When the remote program receives and processes its two segments, it asks for two more segments (see C, in Figure 4-3). The segment counter is incremented from 0 to 2, reenabling data sends. Since PROGA attempted a send that failed, a Link Service Message is delivered to PROGA to indicate that data flow over the link has been reenabled. The Link Service Message is delivered to PROGA when it executes a Receive call when the pending message queue is empty.

A. Remote program requests two segments



B. Local program sends two segments, third send returns an error



C. Remote program requests two segments, local program notified on receive

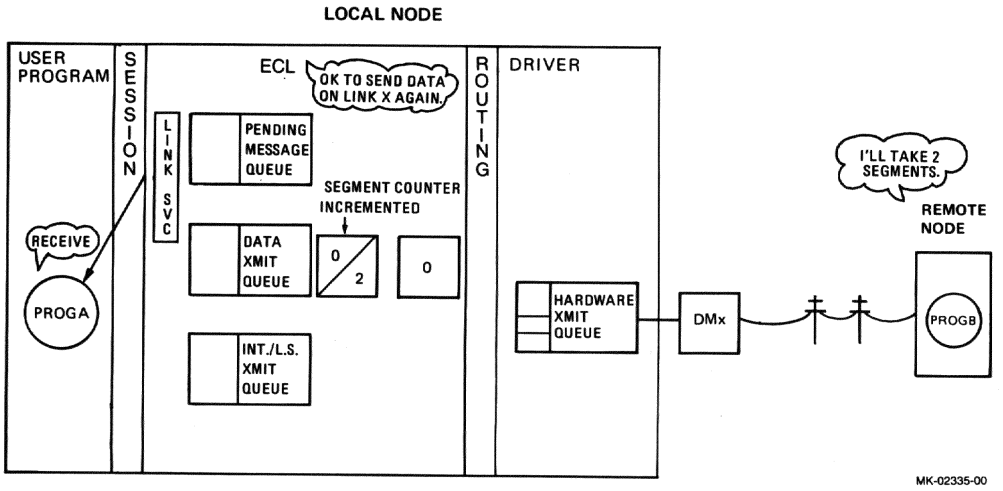


Figure 4-3: How the Flow Control Option Selected by the Remote Program Affects a DECnet/E Program as a Transmitter

Interrupt Message Flow Control

The transmission of Interrupt Messages over a logical link is subject to flow control by a request count mechanism.

All logical links are initiated with an outstanding Interrupt Message request count of one from the remote system. If the local program sends an Interrupt Message over the logical link, this count is decremented to zero. The local ECL does not allow another Interrupt Message to be sent over this logical link until the remote ECL or remote program sends a Link Service Message, incrementing the request count. An attempt by the local program to send an Interrupt Message while the counter is zero results in an error.

Once the request count is incremented by the remote system or program, the local ECL reenables Interrupt Message flow. If the local program has tried to send an Interrupt Message while the count was zero, ECL notifies the local program with a Link Service Message that Interrupt Messages can again be sent over this logical link. This Link Service Message is delivered to the local program on a Receive call when the pending message queue is empty. If the local program has not tried to send an Interrupt Message over the link while flow was disabled, no Link Service Message is delivered.

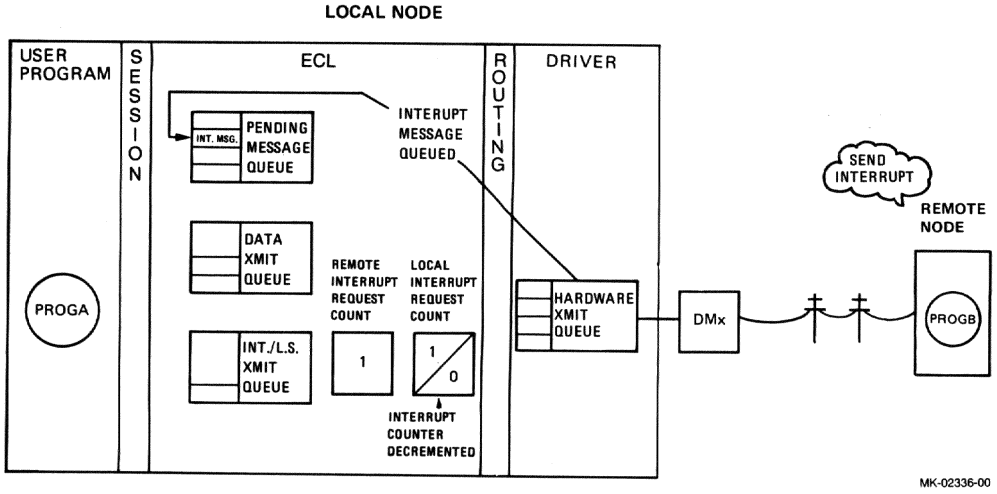
On the other hand, DECnet/E initiates all logical links with an outstanding request for one Interrupt Message. If the remote program sends an Interrupt Message over the link, the local ECL decrements this count to zero. The local ECL cannot accept another Interrupt Message until the local program requests one by sending a Link Service Message to increment the count. This action also increments the request counter at the remote end and thus allows the remote program to send an Interrupt Message, if desired. DECnet/E Session Control does not allow the local program to request an Interrupt Message for a logical link unless a previous Interrupt Message on the link has been received.

In DECnet/E, only one Interrupt Message can be requested on a logical link at any one time. That is, the request count is either zero or one. If a program tries to issue a Link Service Message requesting an Interrupt Message on a link when the request count is already one, the Link Service Message results in an error.

Figure 4-4 shows Interrupt Message flow control for a DECnet/E receiver. The remote program sends an Interrupt Message over a logical link (see A, in Figure 4-4). The local ECL decrements the Interrupt Message counter to zero and does not accept further Interrupt Messages over this link. The remote ECL also decrements its own counter and inhibits further Interrupt Messages from being sent. The local ECL maintains its counter as a safeguard measure. The local program sends a Link Service Message requesting one Interrupt Message over this logical link (see B, in Figure 4-4). The local ECL

increments its counter and tells the remote ECL to reenable Interrupt Messages over the link. The local program must receive the first Interrupt Message before the local ECL can send Link Service Message.

A. Remote program sends an Interrupt Message over a logical link



B. Local program reenables interrupts for the link (allowed only when previous Interrupt Message for this logical link has been received)

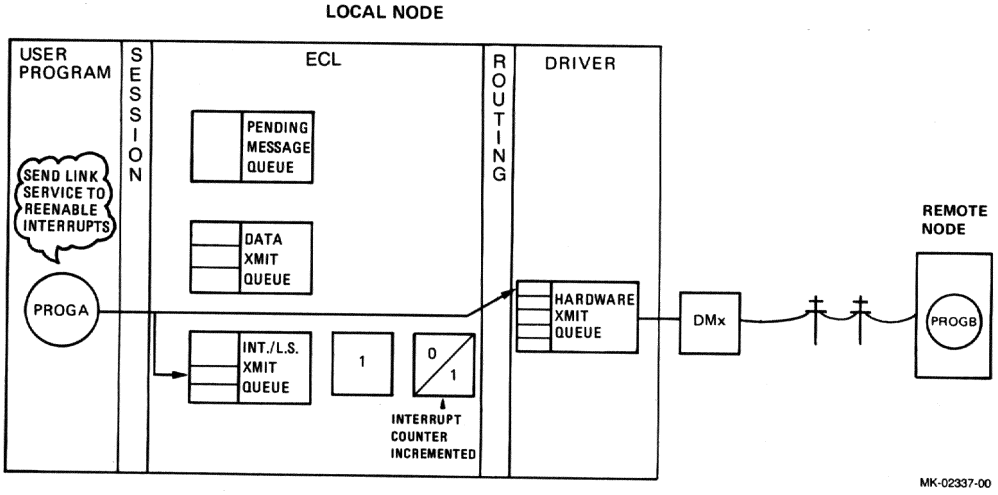


Figure 4-4: How Interrupt Message Flow Control Affects a DECnet/E Program as a Receiver

Programming Hints for Flow Control

A DECnet/E program can be prevented from transmitting for a number of reasons:

- No buffer space is available at the local node. (Prohibits sending of all message types.)
- The Data Transmit Queue is full. (Prohibits sending of Data Messages.)
- The Interrupt/Link Service Transmit Queue is full. (Prohibits sending of Interrupt and Link Service Messages.)
- Message flow over the link has been inhibited by the remote system because of a backpressure condition. (Prohibits sending of Data Messages.)
- No requests are outstanding for segments or logical messages from the remote program. (Prohibits sending of Data Messages.)
- No requests are outstanding for Interrupt Messages. (Prohibits sending of Interrupt Messages.)

The first three conditions indicate congestion at the local node. They are temporary and no notification is given to the program when the condition clears. When one of these errors occurs on a send call, the program should reissue the call after a delay of two to five seconds.

However, the last three conditions indicate congestion at the remote node. Since there is no way to predict how long it can take for one of these conditions to clear, the local ECL does not allow further sends of Data or Interrupt Messages until the remote system indicates that the condition has cleared.

When sending is blocked for one of these last three conditions, any attempt to send a message by the local program results in an error. However, the program can still receive messages. If an attempt to send a message resulted in an error in this fashion, ECL notifies the program when the condition clears. It does this by delivering a Link Service Message to the program when the program issues a Receive call and the pending message queue is empty. No notification is given if the local program did not try to send.

Thus a program whose main function is to transmit data can be designed to send data over a logical link until transmission is blocked by one of the last three conditions. When sending is inhibited, the program can issue Receive calls to process all pending messages. When the pending message queue is empty and all conditions that block flow have

cleared, a Link Service Message is delivered on a Receive call, giving current status information on flow control for the logical link. If more than one logical link has been blocked, Receive calls should be issued for all links until no messages are returned, since link service information can also be available on the other links.

A program whose main function is to receive data is concerned with keeping the pending message queue emptied so the queue does not fill and cause the local ECL to inhibit incoming message flow over the program's logical links. Such a program can use the Receive call with sleep effectively so it is notified when a message is delivered to the queue. In this case, a second Receive call must be issued to retrieve the message.

An interactive program can choose between these mostly send and mostly receive approaches, according to the requirements of the application. A program can also take advantage of the link status information that can be returned after sending Network Data, Interrupt, and Link Service Messages to regulate send and receive requests. The link status information returned is in the same format as a received Link Service Message.

PART II

PROGRAMMING INFORMATION

Chapter 5

Network Programming in BASIC-PLUS

This chapter presents the specific system calls, with detailed formats, used to access the DECnet/E message services in the BASIC-PLUS programming language.

Programming Background

In BASIC-PLUS, the system calls for message services, both local and DECnet, are coded as SYS calls.

General Format of SYS Calls

The general format of the message SYS calls in BASIC-PLUS is:

```
v$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(s%) + ...)
```

where:

v$	is the name of a string that receives control information returned by some of the calls (the target string). The dollar sign ($) indicates a character string variable.
=	is the assignment operator.
SYS(...)	indicates a SYS call.
+	is the concatenation operator required between function, subfunction, and argument values.
CHR$(6%)	is the system function code for a call to the File Processor (FIP). The CHR$ function converts the integer value 6 to the character string format required in SYS calls. The percent sign (%) indicates an integer variable.

CHR$(22%) is the Send/Receive function code.

CHR$(s%) is the user-specified subfunction code. For example, s = 1 indicates a Declare Receiver SYS call; s = -2 indicates a Send Connect Initiate Message SYS call.

... indicates other arguments required for the various system calls. You must specify these arguments in character string format. You can use the CHR$(x%) construction for one-byte integer values. Use CHR$(x%) + CHR$(SWAP%(x%)) for two-byte integers. Enclose ASCII values in quotation marks. Chapter 7 of the *RSTS/E Programming Manual* describes coding conventions for SYS calls.

Buffer Allocation for Message Operations

Local and DECnet communication normally requires the use of buffer areas to hold user data being sent to another program or to receive data from another program. The send calls allow the data to be sent from a buffer or appended to the SYS call string. Buffers are usually required to receive data.

Opening any device on any I/O channel allocates a buffer for I/O to or from the device. There are 12 I/O channels available to each BASIC-PLUS program running under RSTS/E, referenced by the numbers 1 through 12. A convenient way to allocate a buffer for message operations is to open the null device on any channel, using the RECORDSIZE option to specify the length of the buffer desired. For example, to allocate one 128-byte buffer:

```
OPEN "_NL:" AS FILE 12%, RECORDSIZE 128%
```

The null device is used because it is always available and can be opened as many times as necessary (up to 12) to obtain required buffers.

In the Send/Receive SYS calls, you can reference the buffer by the channel number, CHR$(12%) in the previous example. You can also use channel 0 if 128 is added to the channel number, for example, CHR$(128%+0%). (This, in fact, works for all channel numbers: CHR$(128%+x%), for x = 0 to 12.) You can reference data in the buffer with FIELD statements or move this data with standard LSET and RSET statements in BASIC-PLUS.

Notation Used in This Chapter

The SYS call descriptions in this chapter are organized as separate sections. The heading gives the name of the call and the word "Local" or "DECnet" to indicate whether the call is used for local or network message services.

The first page of each call description consists of an introductory paragraph, followed by a table showing the specific format of the data passed and returned for each call. The first column of the table indicates the byte position of each field within the string. The second column indicates the value of the field, and the third column gives a brief description of the way the value is used. Lowercase items in italic are to be supplied by you. The following forms appear:

CHR$(6%)	indicates a one-byte field, in character string format, that must have the value shown (6 in this particular case).
CHR$(x%)	indicates a one-byte field, in character string format, with x being an integer constant or variable name supplied by you.
CHR$(0%) or 0%	indicates that the integer value zero appears, in character string format, in all bytes of the field. Zeros must appear in reserved fields. You can code as CHR$(0%) for one-byte fields or as STRINGS$(n%,0%) for n-byte fields.
x%	indicates a two-byte field, with x being an integer constant or variable supplied by the user. You can convert such variables to character string format using the construction CHR$(x%) + CHR$(SWAP%(x%)).
abc$	indicates a character string value; either a constant enclosed in quotes (for example, "Payroll Summary") or a variable name that has been previously defined to contain the desired string value.

The format table is usually followed by an expanded discussion of the user-supplied arguments. A description of possible errors that can occur for each call is given next, often followed by a brief example of the call.

Declare Receiver (Local and DECnet)

You must execute the Declare Receiver SYS call before a program can receive any messages or send any network messages. However, you can send local messages without first executing this call. The call defines an identifying object type or name (or both) and any restrictions on incoming messages. The Monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block (RIB) for the job (see the section "DECnet/E Use of the Declare Receiver Call" in Chapter 3). It remains in effect until you issue a Remove Receiver call for the job.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(1%)	Declare Receiver subfunction code.
4	CHR$(0%)	Reserved -- must be zero.
5-10	*name*$	The receiver name. The receiver name must be a one- to six-character ASCII string. It must be left-justified and padded to six characters with spaces. The receiver name must contain only printing ASCII characters (characters with ASCII decimal values in the range 33 to 126) and cannot contain leading or embedded spaces. However, you can specify a blank receiver name (six spaces). If you do not have SYSIO privilege and you specify a nonblank name, it must contain six characters and the last two characters must be your job number, as two ASCII digits.
11-20	0%	Reserved -- must be zero.
21	CHR$(*objtyp*%)	The object type code. If the value is 0, the receiver may not be referred to by object type. If the value is not 0, the receiver may be either a local or network object type, depending on if the program asks for incoming logical links (n%).

Data Passed (Cont.)

Bytes	Value	Description
22	CHR$(*access%*)	Access control field = *l%* + *p%* + *n%* + *o%* + *s%*

This byte controls the types of senders that are allowed to queue messages for this job and also controls system handling of queued messages. It is the sum of the following five bit values:

l% Local/No local senders

If l%=0%, messages from local senders are not queued. Local senders who use network functions are considered network senders in this context.

If l%=1%, messages from local senders are queued.

p% Local Privileged/Local Nonprivileged

This bit is ignored if l%=0% (no local senders allowed).

If p%=0%, local senders without the SEND privilege can queue messages.

If p%=2%, local senders must have the SEND privilege.

n% Network Logical Links/No Logical Links

This bit controls the queuing of requests for DECnet/E logical links. For local interjob communication, this bit should be zero.

If the receiver does not have SYSIO privilege, this bit must be zero.

o% Network Single Links/No Single Links

This bit controls the queuing of single network links. For local interjob communication, this bit should be zero.

s% Sleep/No Sleep

If s%=0%, pending messages block execution of a conditional SLEEP. (In a conditional SLEEP, the

Data Passed (Cont.)

Bytes	Value	Description
		monitor checks for certain conditions before executing the SLEEP statement. One of these conditions is pending messages in the job's message queue. If s%=16%, pending messages do not block execution of a conditional SLEEP.
23-24	0%	Reserved -- must be zero.
25	CHR$(*mmax%*)	The message maximum. The message maximum can be any value between 1 and 255. However, if the job does not have EXQTA privilege, this value cannot exceed the value in the account's second quota block. See Discussion.
26	CHR$(*lmax%*)	Inbound link maximum. The incoming link maximum declares the maximum number of incoming DECnet logical links a job will support at any one time. For local interjob communication, the incoming link maximum should be 0. If you have SYSIO privilege, the incoming link maximum can have any value between 0 and 255. If the receiver does not have SYSIO privilege, this value must be zero.
27-28	0%	Reserved -- must be zero.
29	CHR$(*omax%*)	Outbound link maximum. DECnet uses this parameter as the number of outgoing links. If you have EXQTA privilege, the outbound link maximum can have any value between 0 and 255. If you do not have EXQTA privilege, this value must be 0 or 1. (Note that in either case, a value of 0 is interpreted as a default to the maximum value.)
30	0%	Reserved -- must be zero for all receivers except an EMT logger.
31-34	0%	Reserved -- must be zero.

Data Passed (Cont.)

Bytes	Value	Description
35	CHR$(*rib%*)	Receiver ID (RIB) block number. It must be a value between 0 and 255; values 128 through 255 are reserved for use by DIGITAL. A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's second quota block.
36-40	0%	Reserved -- must be zero.

Data Returned

No meaningful data is returned.

Discussion

name

The logical name field is a string containing 1 to 6 characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name is left-justified in the string and (if necessary) padded to six characters with spaces. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), the string should consist of six spaces.

The name identifies the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the network object type code, if one is declared.

If the logical name is null, only network access by network object type is allowed (see discussion of *name*, *objtyp*, and *access*). Otherwise, the logical name must be nonnull and unique.

objtyp

If you have not requested incoming logical links, then a nonzero object type here is interpreted as a local object type. See the *RSTS/E Programming Manual* for more information. If you have requested incoming logical links, then a nonzero object type is interpreted as a network object type.

A network object type code is another form of network addressing (see the section "Declaring Identity" in Chapter 3). The network object type defines some service that the program performs. If the calling

program is addressed by object code alone (*name* is null and *access* indicates network access only), multiple copies of the program can be executed simultaneously. Unlike *name*, the object type code need not be unique -- multiple copies of a program can declare their identity with the same type code simultaneously. If *name* is null, *objtyp* cannot be zero (see the section "Network Addressing" in Chapter 4).

Acceptable values for the object type code range from 0 through 255. The value 0 indicates that object code addressing is not being used for the calling program. The range from 1 to 127 is reserved for DIGITAL use (see Appendix A). The range from 128 to 255 is available for definition and use by a network installation.

access

This field contains the sum of five bit values (*l*% + *p*% + *n*% + *s*% + *o*%) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values (*l*, *p*, and *n*) limit the types of senders that DECnet/E lets queue messages for the calling program. However, they do not limit the kind of messages that can be sent.

If *l* = 0, messages from local senders are not queued for the calling program. (Local senders who use the network functions are considered network senders in this context.) If *l* = 1, messages from local senders are queued.

If *p* = 0, incoming messages from both local senders with send privilege and without send privilege are queued for the calling program. If *p* = 2, incoming messages from local senders are queued only if the sender has SEND privilege. (This bit is ignored if *l* = 0, that is, if no local senders are allowed.)

If *n* = 0, Session Control rejects incoming requests for logical links. If *n* = 4, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access.

Table 5-1 summarizes access code values and the access permitted for each.

Table 5-1: Types of Receiver Access

Access Code (l%+p%+n%)	Network Logical Links	Local Senders with Privilege	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit value *o* is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If *o* = 0, incoming requests for logical links do not affect the program's link maximum. However, if *o* = 8, after queuing an incoming Connect Initiate Message to the receiver, Session Control sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 8, a program prevents the possibility of a Connect Initiate Message being queued to it after its one logical link is disconnected and before it issues a Remove Receiver call. For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote system, Session Control queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that the first link is broken and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value *s* is used to modify the function of the RSTS/E conditional Sleep monitor call. If *s* = 0, any unreceived messages queued for the program blocks the execution of a conditional Sleep call. This is the normal RSTS/E function. However, if *s* = 16, the monitor does not

check the program's message queue when determining whether or not it should suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can still block the sleep, but a pending message does not.

name, *objtyp*, and *access*

Session Control checks the values of the *name*, *objtyp*, and *access* fields to make sure that they are consistent. For example, a null logical name and an access code allowing local senders would be inconsistent since local senders can address the program by logical name. The system validates these arguments as shown in Table 5-2. If the arguments are invalid, the system rejects the Declare Receiver call, returning an error (ERR = 18). If the arguments are valid, the system allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (ERR = 4). A retry may be succeed.

Table 5-2: System Validation of Name, Object, and Access Parameters

Access Code (l%+p%+n%)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local (only)	Ignored	Must be nonnull and unique.
4,6	Network (only)	Zero	Must be nonnull and unique.
		Nonzero	If *name* is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist.
			If *name* is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.

Table 5-2: System Validation of Name, Object, and Access Parameters (Cont.)

Access Code (l%+p%+n%)	Senders Permitted	Object Type	Logical Name
5,7	Network or Local	Any value	Must be nonnull and unique so local senders can refer to the calling program by logical name. Network senders can use the logical name or object type (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

mmax

The message maximum (*mmax*) limits the number of messages that are queued for the calling program. The value can range from 0 to 255.

Session Control keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the message maximum set by the receiver:

- Incoming Local Data Messages are no longer queued for the receiving program. An error is returned to the local sender.
- Incoming Connect Initiate Messages cause Session Control to determine if another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
- An incoming Network Data Message causes ECL to inhibit incoming data messages on that particular logical link. See the section "Backpressure Flow Control" in Chapter 4 for more details.

lmax

The link maximum (*lmax*) limits the number of incoming requests for logical links that are queued for the calling program. The value can range from 0 to 63.

If the number of links currently active is greater than or equal to *lmax* when a remote Connect Initiate Message is received for the

program, the local Session Control determines if another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting *n* = 0 in the access control field. That is, Session Control rejects incoming requests for logical links. The *lmax* parameter does not limit the number of logical links initiated by the program. That is, it does not stop the program from sending Connect Initiate Messages.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL. You specify the RIB number in byte 35; if you do not want to use multiple RIBs, set this byte to zero.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Multiple RIBs can be useful in a program that performs several functions. For example, in certain applications, you might want to use one RIB for high priority messages or messages from specially privileged senders, and another for lower priority messages or messages from any sender. If the program consists of several subroutines or modules, each one can process messages independently using different RIBs.

Possible Errors

ERR value	Meaning
2	ILLEGAL FILE NAME

One of the following occurred:

- The receiver name passed in bytes 5-10 contains nonprinting ASCII characters or leading or embedded spaces.
- A job that did not have SYSIO privilege passed a nonblank receiver name that does not contain its job number in bytes 9 and 10.
- The specified local object type code is invalid.

3 ACCOUNT OR DEVICE IN USE

The calling job already exists in the receiver's list of declared receivers for the specified receiver ID block (RIB) number. This error may indicate that the program contains a logic error or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the last case, the calling job should remove itself (with a Remove call, see the section "Remove Receiver").

10 PROTECTION VIOLATION

One of the following occurred:

- The specified RIB number is out of range.
- The caller does not have sufficient privilege at the time certain functions were attempted in the receiver declaration. (For example, a caller needs SYSIO privilege to declare "single-instance" local object type.)

16 NAME OR ACCOUNT NOW EXISTS

The receiver name passed in bytes 5-10 is already being used as a receiver ID (this error cannot occur if the name specified is blank), or a specified "single-instance" local object type is already in use.

Possible Errors (Cont.)

ERR value	Meaning
18	ILLEGAL SYS( ) USAGE The receiver name, object type, and access parameters passed are inconsistent.
31	ILLEGAL BYTE COUNT FOR I/O The value you specified in byte 30 is out of range (refers to an EMT logger).
32	NO BUFFER SPACE AVAILABLE When the job attempted to declare itself as a receiving job, there were no small buffers available for the declaration arguments. Since the system's use of small buffers is dynamic, a retry may succeed.
66	?MISSING SPECIAL FEATURE The call you attempted requires an optional feature (such as EMT logging) that is not available on your system.

Example

The following example shows a receiver declaration allowing only network access using logical links. The logical name is ADMIN and no object type code is used. The message maximum (MMAX) is set to 10. If more than 10 messages accumulate in the program's pending message queue, an incoming Connect Initiate Message causes Session Control to send back a Connect Reject Message to the remote program and any further incoming Data Messages cause Session Control to inhibit incoming message flow on all the program's logical links because of the backpressure condition. Likewise, the link maximum (LMAX) is set to 1.

When the program establishes one logical link, an incoming Connect Initiate Message causes Session Control to send back a Connect Reject Message to the remote program. Receiver ID block #3 is used.

The program will not attempt to limit outbound links, so 0MAX is zero. For example:

```
LNAME$  = "ADMIN "
ACCESS% = 4%
0MAX%   = 0%
MMAX%   = 10%
LMAX%   = 1%
RIB%    = 3%
!
! DECLARE RECEIVER EXAMPLE
!
X$ = SYS(CHR$(6%)+CHR$(22%)+CHR$(1%)+CHR$(0%)
      +LNAME$ + STRING$(10%,0%)
      +CHR$(0%)
      +CHR$(ACCESS%)
      +CHR$(0%) + CHR$(0%)
      +CHR$(MMAX%)
      +CHR$(LMAX%)
      +CHR$(0MAX%)
      +STRING$(7%,0%)
      +CHR$(RIB%))
```

Remove Receiver (Local and DECnet)

This call removes a receiver from the system's list of declared receivers. You can remove all RIBs for a job or a specific RIB for a job. (Be careful when removing all RIBs for a job.) When this call is executed, all pending messages for the receiver are discarded and all logical links are aborted. You should execute this call when message processing is being terminated but the job is to continue running. This prevents unwanted messages from accumulating in the queue of pending messages.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(0%)	Remove Receiver subfunction code.
4	CHR$(0%) or CHR$(*jobx2*%)	Zero, to remove the calling job (normal usage), or Job number times two, to remove another local RSTS/E job. Only programs with JOBCTL privilege can issue this call with byte 4 ≠ 0. Add 128% to this byte for "conditional" remove.
5-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID block (RIB) number to remove, from 0 to 255.
36	CHR$(0%)	Removes the RIB specified in byte 35. A nonzero value in this byte removes all RIBs for the job specified in byte 4.
37-40	0%	Reserved -- must be zero.

Data Returned

No meaningful data is returned.

Possible Errors

ERR value	Meaning
3	ACCOUNT OR DEVICE IN USE This occurs for a conditional remove if there still are messages pending.
10	PROTECTION VIOLATION The caller does not have JOBCTL privilege and attempted to remove another job (that is, byte 4 is nonzero).
18	ILLEGAL SYS( ) USAGE The number passed in byte 4 was odd. Byte 4 must be zero to remove the calling program or job number times two to remove another job.

Discussion

Note that both the LOGOUT SYS function and the KILL SYS function execute this call with a nonzero value in byte 36.

The "conditional" remove operation is useful for programs that want to process messages until none remain, then remove the receiver. If you do a receive, find that there are no more messages, and then do a remove, a message could arrive in the time between the receive and the remove; the remove operation would discard that message. To avoid this problem, add 128% to byte 4 for "conditional" remove; in this case, the system returns the error ?Account or device in use (ERR=3) rather than discard messages if some messages are still waiting to be received.

Example

The following example shows a typical Remove Receiver call, removing RIB #0 of the current job:

```
!
! EXAMPLE OF REMOVE RECEIVER
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(0%))
```

Get Local Node Parameters (DECnet)

The Get Local Node Parameters SYS call returns information to the calling program concerning the local node's network parameters. Data returned includes the node name or node number, address, and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. See the section "Automatic Job Start-Up" in Chapter 4 for a discussion of the default user's account.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-19%)	Get Local Node Parameters subfunction code.
4	CHR$(1%)	Must be set to "1".
5-40	0%	Reserved -- must be zero.

Data Returned

The local node network parameters are returned to the calling program in the target string. The format of the information is as follows:

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-19%)	Get Local Node Parameters subfunction code.
4	--	Reserved.
5-10	*name*$	Local node name: 1 to 6 characters. Left-justified with space fill to 6 characters.
11-28	--	Reserved.
29-30	*ppn*%	Default project-programmer number: Byte 29 = programmer number, byte 30 = project number. Or both zero if none defined.
31-40	--	Reserved.

Possible Errors

ERR value	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows the Get Local Node Parameters call:

```
!
! EXAMPLE OF GET LOCAL NODE PARAMETERS
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-19%) + CHR$(1%))
```

Log User Event (DECnet)

The Log User Event SYS call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

Note

This SYS call performs a highly specialized function which requires a great deal of special knowledge of DECnet/E and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the experienced network user and is not intended for normal network programs.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-10%)	Log User Event subfunction code.
4	CHR$(*evmod*%)	Error return modifier. If 1, log Missed event message if event queue is full; if 0, return NOROOM status return.
5-6	*evcode*%	Event code representing the class (*evcls*%) and type (*evtyp*%) of the user event. The format of this two-byte integer is as follows: *evcode* = *evcls* * 64% + *evtyp*
7-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the optional user data to be passed to the event logger for processing. If 0, this data is passed as part of the string.
12	CHR$(0%)	Reserved -- must be zero.

Data Passed (Cont.)

Bytes	Value	Description
13-14	*length%*	Length, in bytes, of data to be passed from the I/O buffer, 0 to 200. If 0, the number of bytes equals the buffer size minus the *offset*. Ignored if *chnl* = 0.
15-16	*offset%*	Offset, in bytes, from the beginning of the buffer. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-20	0%	Reserved -- must be zero.
21-22	*enval%*	Identifier of the entity with which this event is concerned.
23	CHR$(*entyp%*)	Type of entity with which this event is concerned.
24-40	0%	Reserved -- must be zero.
41+	*data$*	Optional user data, if sent as part of the string. Maximum of 200 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

evmod

This parameter specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with an error of NOROOM (ERR = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a Missed event message, and the call completes successfully.

Note that in the second case, the user data is not logged. The system merely logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet/E internal events when the event queue is full or there are no more buffers.

evcode

This parameter is a two-byte integer representing the class (*evcls%*) and type (*evtyp%*) of the user event to be logged. The class and type are packed into one two-byte value as follows:

*evcode = evcls * 64% + evtyp*

Events are divided into classes, according to the DIGITAL Network Architecture (DNA) layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480. The event type, *evtyp*, must be in the range of 0 to 31.

enval

This parameter associates a value to the entity specified by the *entyp* parameter. If the entity is a node, set *enval* to the node address. If the entity is a line or a circuit, set *enval* to the address of the Device Data Block (DDB) maintained for the device.

entyp

This parameter specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E:

-1	No specific entity
0	Node
1	Line
3	Circuit

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which channel buffer contains the user data to be processed with the event. Or it can be 0, indicating that this information is appended to the SYS call string.

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. DECnet parameter types can be used if the event concerns a node, a circuit, or a line. In this case, the event logger outputs standard descriptive text for the specific parameter. If a DECnet/E parameter type is not used, parameter types in the range of 3900 to 4095 can be specified. See the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for further information on event definitions and parameters.

Up to 200 bytes of data can be passed to the event logger for processing.

length

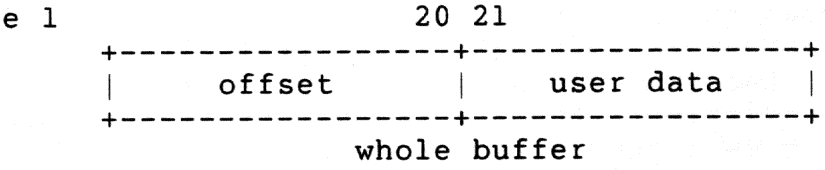
The *length* parameter indicates how many bytes are to be processed from the channel buffer. Up to 200 bytes of data can be passed.

If *length* = 0, the number of bytes processed is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes passed to the event processor is equal to the buffer size minus *offset*. This size must also be less than or equal to 200.

The *length* parameter is ignored if *chnl* = 0; that is, if the data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data is assumed to be in the buffer in contiguous bytes. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

The event data to be passed to the event logger for processing can be included as part of the SYS call string, rather than in an I/O buffer. If this is the case, the data is passed in these bytes.

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod*=0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.

Possible Errors (Cont.)

ERR value	Meaning
10	PROTECTION VIOLATION The calling program did not have SYSIO or SEND privileges at the time the Log User Event system call was issued.
18	ILLEGAL SYS( ) USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows the logging of a user event, class 480. and type 3. The entity node number is 135. The optional data passed for processing is part of the SYS call string and consists of an *11a* parameter number (2130) and value. A Missed event message is logged if the event itself cannot be logged. For example:

```
EVMOD% = 1%
EVCLS% = 480%
EVTYP% = 3%
EVCODE% = EVCLS% * 64% + EVTYP%
ENTYP% = 0%
ENVAL% = 135%
PRMTYP% = 2130%
DATTYP% = 2%
DATVAL% = 40960%
DATA% = CHR$(PRMTYP%) + CHR$(SWAP%(PRMTYP%)) + CHR$(DATTYP%)
        + CHR$(DATVAL%) + CHR$(SWAP%(DATVAL%))
!
! EXAMPLE OF LOG USER EVENT CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-10%)
      + CHR$(EVMOD%)
      + CHR$(EVCODE%) + CHR$(SWAP%(EVCODE%))
      + STRING$(14%,0%)
      + CHR$(ENVAL%) + CHR$(SWAP%(ENVAL%))
      + CHR$(ENTYP%)
      + STRING$(17%,0%)
      + DATA$)
```

The Log User Event SYS call in the previous example generates the following event message:

```
Event type 480.3
Occurred 24-Sep-86 15:31:39.7 on node 3.135 (EDERE)
Node address 3.135 (EDERE)
Parameter #2130 = 40960
```

Send SYS Calls

There are ten separate send SYS calls -- two local and eight network. The network calls cause Session Control to format a message that is then either queued to a local program or transmitted across the network to a remote program.

Send Local Data Message (Local)

The Send Local Data Message SYS call sends up to 532 bytes of user data to another program in the local RSTS/E environment. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege. The receiving program is identified in the call by either its job number or by the logical name with which it declared itself a receiver.

Up to 512 bytes of user data can be sent from a channel buffer or appended to the string. An additional 20 bytes of user data can be sent from the string.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-1%)	Subfunction code.
4	CHR$(*job*%)	If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. See the Discussion for valid LOTs. Otherwise, *job*% can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.
5-10	*name*$	Declared logical name of local program to receive the message: 1 to 6 characters. Left-justified with space fill to 6 characters. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

Data Passed (Cont.)

Bytes	Value	Description
11	CHR$(*chnl*%)	Channel number of buffer containing the message data, 1 to 12. If 0, the data is to be sent as part of the string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length, in bytes, of the message data to be sent from the buffer, 1 to 512. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset, in bytes, from the beginning of the buffer where the message data begins. If = 0, the message data is assumed to start at beginning of the buffer. Ignored if *chnl* = 0.
17-20	0%	Reserved -- must be zero.
21-40	*param*$	See discussion.
41+	*data*$	0 to 512 bytes of message data to be sent to the local program, if sent as part of the string (that is, if *chnl* = 0). Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

job

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate)

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number times 2 of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data

passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The logical name identifies the intended local receiver when *job* = 0. The intended receiver must have declared its identity with this name in its Declare Receiver call. The name consists of one to six characters, left-justified in the field and, if necessary, padded to six characters with spaces. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). Leading or embedded spaces are invalid. This field is ignored when *job* ≠ 0.

chnl

The *chnl* parameter is either an integer from 1 to 12, indicating which I/O buffer contains the message data, or the value zero, indicating that the message data is part of the SYS call string.

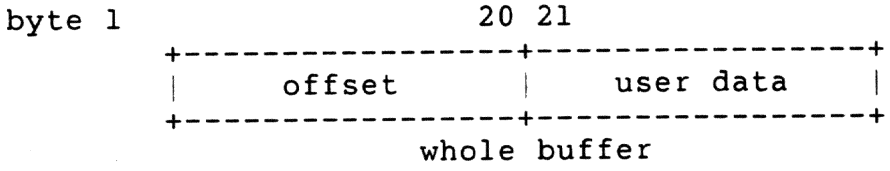
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. The maximum value for *length* is 512. If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement that allocated the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range of 1 to 512.

The length field is ignored if *chnl* = 0, that is, if the user message data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with the *offset* parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

param

The optional *param* string allows up to 20 additional bytes of user data to be passed. It is delivered to the local program separate from the message data. For example, when a BASIC-PLUS program issues a Receive call that returns a Local Data Message, the parameter data is returned in the target string for the Receive call and the message data is returned in the channel buffer specified

data

The user data (if any) can be sent from the SYS call string (*data*) or from the I/O buffer specified by the *chnl* parameter. Up to 512 bytes can be sent in either case. If a buffer is used, remember that *length* bytes are transmitted and that the first *offset* bytes in the buffer are not sent.

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The calling program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 of the data passed is not open. The program must OPEN the channel to allocate a buffer for message operations.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	ILLEGAL SYS( ) USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.

Possible Errors (Cont.)

ERR value	Meaning
31	ILLEGAL BYTE COUNT FOR I/O

The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true:

- The *offset* must be less than the buffer size.
- The *length* must be less than or equal to the buffer size minus *offset*.
- The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be less than or equal to 512.

The *offset* and *length* fields are checked for validity whenever *chnl* (byte 11) ≠ 0.

32	NO BUFFER SPACE AVAILABLE

System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Example

This example shows a local send to a program named HARVEY. The user data is sent from the SYS call string, beginning in byte 41:

```
MSG$ = "THIS MESSAGE IS SENT FROM THE SYS CALL STRING"
!
!  EXAMPLE OF LOCAL DATA SEND
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-1%) + CHR$(0%)
      + "HARVEY" + STRING$(30%,0%)
      + MSG$)
```

Send Local Data Message with Privilege Mask (Local)

The Send Local Data Message with Privilege Mask sends a data message plus a privilege mask supplied by the monitor. This subfunction provides a method for a program to tell another program about a job's current privileges and guarantees that the data cannot be falsified.

A local job can send its privileges to a declared receiver in the same way as the Send Local Data Message call. See the discussion in the previous section.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-11%)	Subfunction code.
4	CHR$(*job*%)	If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. See the Discussion for valid LOTs. Otherwise, *job*% can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.
5-10	*name*$	Declared logical name of local program to receive the message: 1 to 6 characters. Left-justified with space fill to 6 characters. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.
11	CHR$(*chnl*%)	Channel number of buffer containing the message data, 1 to 12. If 0, the data is to be sent as part of the string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length, in bytes, of the message data to be sent from the buffer, 1 to 512. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.

Data Passed (Cont.)

Bytes	Value	Description
15-16	*offset%*	Offset, in bytes, from the beginning of the buffer where the message data begins. If = 0, the message data is assumed to start at beginning of the buffer. Ignored if *chnl* = 0.
17-28	0%	Reserved -- must be zero.
29-40	*param$*	See discussion.
41+	*data$*	0 to 512 bytes of message data to be sent to the local program, if sent as part of the string (that is, if *chnl* = 0). Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

job

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate)

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number times 2 of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The logical name identifies the intended local receiver when *job* = 0. The intended receiver must have declared its identity with this name in its Declare Receiver call. The name consists of one to six characters, left-justified in the field and, if necessary, padded to six characters with spaces. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). Leading or embedded spaces are invalid. This field is ignored when *job* $\neq$ 0.

chnl

The *chnl* parameter is either an integer from 1 to 12, indicating which I/O buffer contains the message data, or the value zero, indicating that the message data is part of the SYS call string.

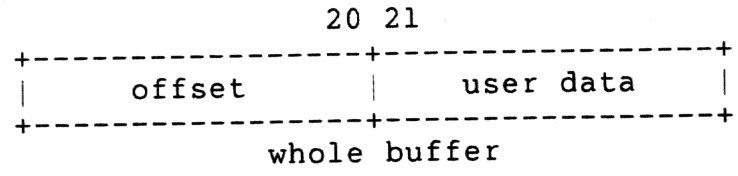
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. The maximum value for *length* is 512. If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement that allocated the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range of 1 to 512.

The length field is ignored if *chnl* = 0, that is, if the user message data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with the *offset* parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

param

The optional *param* string allows up to 12 additional bytes of user data to be passed. It is delivered to the local program separate from the message data. For example, when a BASIC-PLUS program issues a Receive call that returns a Local Data Message, the parameter data is returned in the target string for the Receive call and the message data is returned in the channel buffer specified

data

The user data (if any) can be sent from the SYS call string (*data*) or from the I/O buffer specified by the *chnl* parameter. Up to 512 bytes can be sent in either case. If a buffer is used, remember that *length* bytes are transmitted and that the first *offset* bytes in the buffer are not sent.

Possible Errors

This call returns the same errors as the Send Local Data Message call. See the previous section.

Send Connect Initiate Message (DECnet)

The Send Connect Initiate Message SYS call requests a logical link to another program. The other program is identified with a Connect Data Block and the User Link Address which the calling program refers to when the link is established. You can also specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 bytes of optional user data can be passed to the other program.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-2%)	Send Connect Initiate Message subfunction code.
4	CHR$(*ula*%)	User Link Address, 1 to 255.
5-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the Connect Data Block and optional user data, 1 to 12. If 0, these are to be sent as part of the string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length, in bytes, of information to be sent from the I/O buffer (including the Connect Data Block), 172 to 188. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset, in bytes, from the beginning of the buffer where the Connect Data Block begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-22	0%	Reserved--must be zero.
23	CHR$(*llmod*%)	Local link modifier. Indicates the type of flow control: 0 = None 1 = Segment 2 = Message
24	CHR$(0%)	Reserved -- must be zero.

Data Passed (Cont.)

Bytes	Value	Description
25-26	*rmax%*	Maximum amount of user data (in bytes) that the local program accepts in one Network Data Message from the remote program.
27-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.
36-40	0%	Reserved -- must be zero.
41-212	*cdb$*	The Connect Data Block, if sent as part of the string (if *chnl* = 0). Ignored if *chnl* ≠ 0. (In either case, the Connect Data Block must equal 172 bytes.)
213+	*data$*	Optional user data, if sent as part of the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

ula

The User Link Address (*ula*) can be an integer from 1 to 255. This value is used in later calls to refer to this link (assuming that the link being requested by this call is accepted by the remote ECL and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which channel buffer contains the Connect Data Block and the (optional) message data. Or it can be 0, indicating that this information is appended to the SYS call string.

length

The length parameter indicates how many bytes, including the Connect Data Block, are to be sent from the channel buffer. Since the Connect

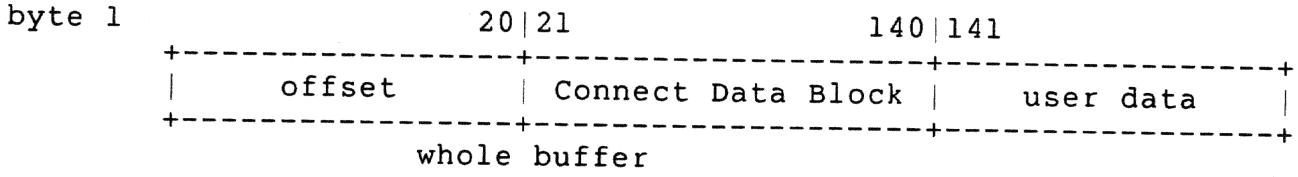
Data Block is 172 bytes long and the optional message data can be from 1 to 16 bytes long, acceptable values for *length* in this request run from 172 to 188; or it can be 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset* (which must still be in the range 172 to 188).

The *length* parameter is ignored if *chnl* = 0; that is, if the Connect Data Block and message data are appended to the SYS call string.

offset

If an I/O buffer is used, the Connect Data Block and message data are assumed to be in the buffer in contiguous bytes. They can be offset from the beginning of the buffer, and if so, the size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the Connect Data Block begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

llmod

The *llmod* value indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 = No flow control
1 = Segment flow control
2 = Message flow control

Chapter 4 describes flow control in detail. Remember that flow control is independent for each end of a logical link. What is requested here applies only to this end of the link. If *llmod* is nonzero, the calling program must issue Link Service calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces can impose a limit on the amount of user message data it is willing to receive on a logical link. See the section "Receive System Call" for a discussion of the *length* parameter for the Receive call. The *rmax* value specifies this limit in bytes.

The size of the receive buffers allocated by ECL limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value. Note that the calling program is informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program which must limit the amount of the user message data it transmits over the logical link according to this value.

connect data block

You can specify the Connect Data Block identifying the remote node and program as part of the SYS call string (*cdb*) or you can give it in the I/O buffer (*chnl*≠0). In either case, it must be 172 bytes long. The format of the 172-byte Connect Data Block is shown in Figure 5-1 and described in detail in Table 5-3.

data

Optional user data (up to 16 bytes) can be given following the Connect Data Block as part of the SYS call string (*data*) or following the Connect Data Block in the I/O buffer (*chnl*≠0).

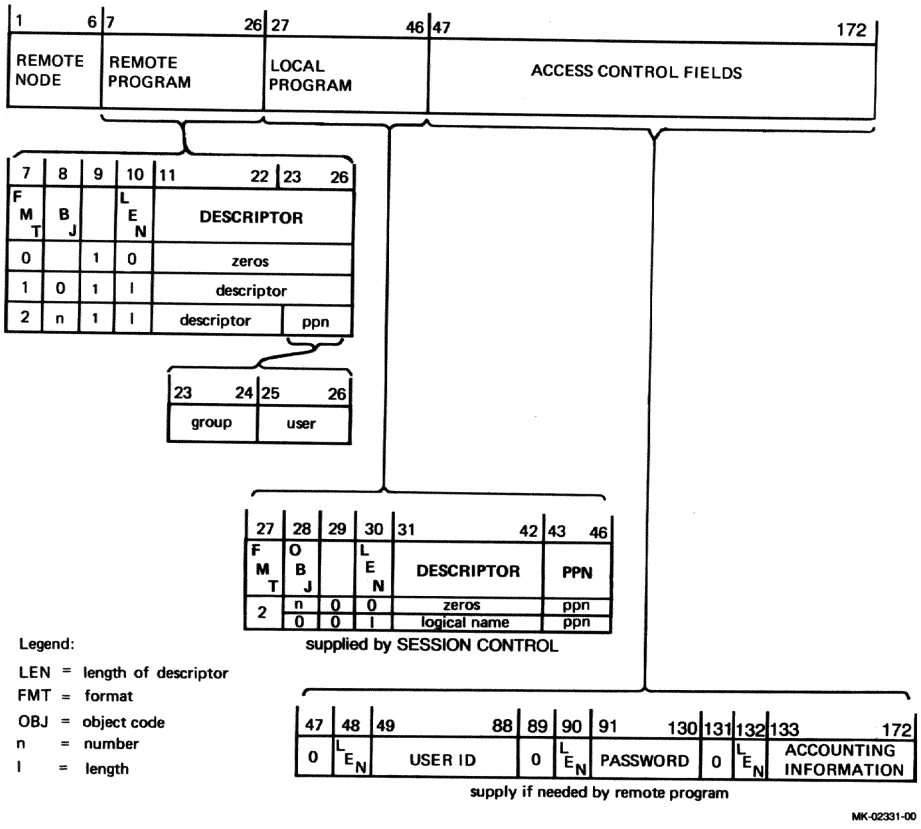


Figure 5-1: Format of Connect Data Block on Send Connect Initiate Message

Table 5-3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node to which the Connect Initiate Message is to be sent. Names shorter than 6 characters must be left-justified with space fill to 6 characters. Leading and embedded spaces are invalid. If no node name is specified (that is, the request is directed to a program at the local node), these 6 bytes contain an ASCII 0 and 5 spaces.
7-26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats: **Format 0** The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9	Must be one.
10-26	Zero (Not used).

Table 5-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	Format 1 The remote program is identified by a descriptor. The fields are: **Bytes** — **Content** 7 — One, to indicate a format 1 name. 8 — Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). 9 — Must be one. 10 — Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field. 11-26 — Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations. **Format 2** This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

Table 5-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The subfields are:
	Bytes / **Content**
	7 — Two, to indicate a format 2 name.
	8 — Object type code of the remote program, if applicable. If zero, the descriptor must be used.
	9 — Must be one.
	10 — Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
	11-22 — Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
	23-24 — Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E PPN.
	25-26 — Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E PPN.
27-46	**Local Program.** This field is filled in by the local session control using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

Table 5-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with an object code alone, the object code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The RSTS/E PPN under which the program is running is passed in bytes 43-46 as the group and user codes: the project number is in byte 43; the programmer number is in byte 45. Both are in binary format.
47-172	**Access Control Fields.** These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number or system password, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described.

Note

Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved -- must be zero.
48	Length of user identification information specified in the next field, 0 to 40 bytes.

Table 5-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
49-88	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, set the length (byte 48) to zero.
89	Reserved -- must be zero.
90	Length of password information specified in the next field, 0 to 40 bytes.
91-130	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, set the length (byte 90) to zero.
131	Reserved -- must be zero.
132	Length of the accounting information specified in the following field, 0 to 40 bytes.
133-172	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.

Possible Errors

ERR value	Meaning
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NOT A VALID DEVICE The node named in the Connect Data Block is not known to the system.
9	I/O CHANNEL NOT OPEN The channel specified by byte 11 of the call is not open. The program must OPEN the channel before using it for message operations.
14	DEVICE HUNG OR WRITE LOCKED The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	TOO MANY OPEN FILES ON UNIT ECL cannot allocate its Local Link Address (*lla*) The maximum number of logical links allowed at the local RSTS/E node (some limit between 1 and 127, as set by the system manager; see the system manager if the limit in effect is inadequate) has been reached. A retry may succeed.

Possible Errors (Cont.)

ERR value	Meaning
18	ILLEGAL SYS() USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Do a Remove Receiver call and reissue the Declare Receiver call.
- An invalid value was detected in one of the following fields:
  - Remote program field in the Connect Data Block.
  - One of the three access control fields in the Connect Data Block.
  - User Link Address (*ula*) given was zero. Valid values range from 1 to 255.
  - The *llmod* value must be 0, 1, or 2.
  - The remote descriptor length (byte 10 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).

22	DISK PACK IS LOCKED OUT

Session Control cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.

Possible Errors (Cont.)

ERR value	Meaning
31	ILLEGAL BYTE COUNT FOR I/O

The *offset* and/or *length* fields passed in bytes 13-16 are invalid. The following relationships must be true:

- The *offset* must be less than the buffer size.
- The *length* must be less than or equal to the buffer size minus the *offset*.
- The number of bytes to be sent (as defined by *length* or if *length* = 0, by buffer size minus *offset*) must be between 172 and 188. This encompasses 172 bytes for the Connect Data Block and a maximum of 16 bytes of data.

The *offset* and *length* fields are checked for validity whenever *chnl* (byte 11) ≠ 0.

32	NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A retry may succeed.

62	NO RUN-TIME SYSTEM

DECnet/E has not been enabled. Normal DECnet calls cannot be executed until DECnet/E is enabled by the system manager.

66	MISSING SPECIAL FEATURE

DECnet/E was not installed. The network functions cannot be executed.

Example

This example shows a Connect Initiate Message directed to a program with object type code 126 at node BOSTON. The Connect Data Block (CDB$) uses a format 0 name to address the remote program. The local program name is supplied by ECL and no access control fields are being used, so the rest of the Connect Data Block is filled with binary zeros to make up 172 bytes.

A ULA of 13 is used, segment flow control is requested by the calling program, and a receive maximum of 256 bytes is specified:

```
ULA% = 13% !USER LINK ADDRESS  = 13
LLMOD% = 1% !SEGMENT FLOW CONTROL REQUESTED
RMAX% = 256% !RECEIVE MAXIMUM SET TO 256 BYTES
CDB$ = "BOSTON" + CHR$(0%) + CHR$(126%) + STRING$(164%,0%)
RIB% = 3% !RIB #3
!
! EXAMPLE OF SEND CONNECT INITIATE
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-2%) + CHR$(ULA%)
      + STRING$(18%,0%) + CHR$(LLMOD%) + CHR$(0%)
      + CHR$(RMAX%) + CHR$(SWAP%(RMAX%))
      + STRING$(8%,0%) + CHR$(RIB%) + STRING$(5%,0%)
      + CDB$ + "TIME-CARD UPDATE")
```

Send Connect Confirm Message (DECnet)

The Send Connect Confirm Message SYS call accepts a remote program's request for a logical link. The call establishes the User Link Address (ULA) that the local program uses to refer to the link and identifies the link being accepted by the Local Link Address (LLA) assigned by the local ECL. The flow control option and receive maximum are specified here and up to 16 bytes of optional message data can be passed to the remote program.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-3%)	Send Connect Confirm Message subfunction code.
4	CHR$(*ula*%)	User link address, 1 to 255.
5-6	*lla*%	Local link address (from bytes 5-6 of received Connect Initiate Message).
7-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the optional message data, 1 to 12. If 0, the data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length, in bytes, of data to be sent from the buffer, 1 to 16. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset, in bytes, from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-22	0%	Reserved -- must be zero.
23	CHR$(*llmod*%)	Local link modifier, used to indicate the type of flow control: 0 = None 1 = Segment 2 = Message

Data Passed (Cont.)

Bytes	Value	Description
24	CHR$(0%)	Reserved -- must be zero.
25-26	*rmax%*	Maximum amount of user data (bytes) that the local program accepts in one Network Data Message from the remote program.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.
36-40	0%	Reserved -- must be zero.
41+	*data$*	Optional message data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

ula

The User Link Address (*ula*) can be an integer from 1 to 255. This value is used in later calls to refer to this link. The *ula* must be unique. That is, one program cannot have two logical links with the same *ula* at the same time.

lla

The Local Link Address (*lla*) identifies the link being accepted. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in bytes 5-6 of the received Connect Initiate Message to which this Connect Confirm Message is a positive response. The section "Format of Received Connect Initiate Message" shows the entire format of a received Connect Initiate Message.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which buffer contains the (optional) user data. Or it can be 0, indicating that this information is appended to the SYS call string.

length

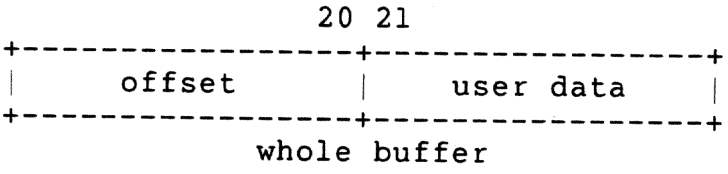
The *length* parameter indicates how many bytes are to be sent from the channel buffer. Acceptable values for *length* in this request can range from 1 to 16, or it can be 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range 1 to 16.

The *length* parameter is ignored if *chnl* = 0; that is, if the user data is appended to the SYS call string.

offset

If an I/O buffer is used, the user data is assumed to be in the buffer in contiguous bytes. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

llmod

The *llmod* value indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 = No flow control
1 = Segment flow control
2 = Message flow control

Chapter 4 describes flow control in detail. Remember that flow control is independent for each end of a logical link -- what is requested here applies only to this end of the link. If *llmod* is nonzero, the calling program must issue Link Service calls to request Network Data Messages from the remote program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces can impose a limit on the amount of user data it is willing to receive in one Network Data Message from the remote program over this logical link. See the discussion of the

length parameter for the Receive call in the section "Receive System Call." The *rmax* value specifies this limit in bytes.

The size of the receive buffers allocated by ECL limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified by *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program which must limit the amount of the user data it transmits in Network Data Messages according to this value.

data

Up to 16 bytes of optional user data can be passed as part of the SYS call string (*data*) or in the channel buffer (*chnl*≠0).

Possible Errors

ERR value	Meaning
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same User Link Address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) specified in bytes 5-6 does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating Connect Initiate Message has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. Since no *ula* was ever established, an *lla* field is available in a received Link Abort Message identifying the link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations (see the section "Buffer Allocation for Message Operations").

Possible Errors (Cont.)

ERR value	Meaning
10	PROTECTION VIOLATION

Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation or rejection results in this error. For example, a second Connect Confirm Message for the same logical link results in this error because the link is already established.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- An invalid value was detected in one of the following fields:
  - *ula* (byte 4) is 0. Acceptable values range from 1 to 255.
  - *llmod* (byte 23) must be 0, 1, or 2.

31 ILLEGAL BYTE COUNT FOR I/O

The *offset* and/or *length* fields passed in bytes 13-16 are invalid. The following relationships must be true:

- The *offset* must be less than the buffer size.
- The *length* must be less than or equal to the buffer size minus the *offset*.
- The number of bytes to be sent as defined by *length* or if *length* = 0, by buffer size minus *offset* must be between 0 and 16.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A retry may succeed.

Possible Errors (Cont.)

ERR value	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, a Send Connect Confirm Message SYS call is issued in response to a previously received Connect Initiate Message whose control information was returned to the target string A$. The MID function places the LLA from A$ in the string LLA$. Other parameters are similar to those in a Connect Initiate Message. For example:

```
LLA$ = MID(A$,5%,2%) !GET LOCAL LINK ADDRESS
ULA% = 25%  !USER LINK ADDRESS = 25
LLMOD% = 2%  !MESSAGE FLOW CONTROL REQUESTED
RMAX% = 512%
RIB% = 3%                !RIB #3
!
! EXAMPLE OF CONNECT CONFIRM
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-3%) + CHR$(ULA%)
      + LLA$ + STRING$(16%,0%)
      + CHR$(LLMOD%) + CHR$(0%)
      + CHR$(RMAX%) + CHR$(SWAP%(RMAX%))
      + STRING$(8%,0%) + CHR$(RIB%))
```

Send Connect Reject Message (DECnet)

The Send Connect Reject Message SYS call rejects a remote program's request for a logical link. The link is identified by the Local Link Address (LLA) established by the local ECL. You can also pass up to 16 bytes of optional user data to the remote program as part of the Connect Reject Message.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-4%)	Send Connect Reject Message subfunction code.
4	CHR$(0%)	Reserved -- must be zero.
5-6	*lla*%	Local link address (from bytes 5-6 of received Connect Initiate Message).
7-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the optional message data, 1 to 12. If 0, the data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length, in bytes, of data to be sent from the buffer, 1 to 16. If 0, the number of bytes sent equals buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset, in bytes, from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-22	0%	Reserved -- must be zero.
23-24	*reason*%	Reason for rejection, if applicable. Can be 0 (normal case for user programs), 34 (access not permitted), or 36 (invalid accounting information).
25-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.

Data Passed (Cont.)

Bytes	Value	Description
36-40	0%	Reserved -- must be zero.
41-56	*data*$	Optional message data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

lla

The Local Link Address (*lla*) identifies the link being rejected. It is a number that the local ECL assigns when it receives a Connect Initiate Message requesting a link. The *lla* is passed to the local program in bytes 5-6 of the received Connect Initiate Message to which this Connect Reject Message is a negative response. The section "Format of Received Connect Initiate Message" shows the entire format of a received Connect Initiate Message.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which buffer contains the (optional) message data to be passed to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

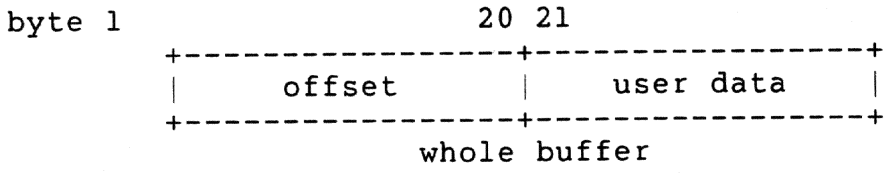
length

The *length* parameter indicates how many bytes of user data, if any, are to be sent from the channel buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This size must also be in the range 1 to 16.

offset

If a channel buffer is used, the user message data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

reason

DECnet/E lets a program specify one of the following three reason codes: 0, 34, or 36.

Most user programs use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The value 34 indicates that the user ID and password subfields of the access control field in the Connect Data Block in the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* ≠ 0).

Possible Errors

ERR value	Meaning
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) given in bytes 5-6 does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating Connect Initiate request has been cancelled. In the latter case, a Disconnect or Link Abort Message has already been queued for this link. Since no *ula* was ever established, an *lla* field is available in the received Link Abort Message identifying the link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations (see the section "Buffer Allocation for Message Operations").
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation or rejection causes this error. For example, a second Connect Reject Message for a logical link that is already established returns this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *reason* code specified in bytes 23-24 was not 0, 34, or 36.

Possible Errors (Cont.)

ERR value	Meaning
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true: o The *offset* must be less than the buffer size. o The *length* must be less than or equal to the buffer size minus the *offset*. o The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Send Connect Reject Message call.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store the user data portion of the Connect Reject Message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, a Connect Reject Message is sent to a remote program in response to a received Connect Initiate Message whose control information was returned to the target string B$ on RIB RIB%:

```
!
!EXAMPLE OF CONNECT REJECT
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-4%) + CHR$(0%)
      + MID(B$,5%,2%) + STRING$(28%,0%)
      + CHR$(RIB%) + STRING$(5%,0%)
      + "TOO LATE")
```

Send Network Data Message (DECnet)

The Send Network Data Message SYS call transmits user data over an established logical link. The amount of user data you can transmit in one Network Data Message is called a segment. You can set a flag to indicate that the segment is also the end of a logical message. If the remote program requested flow control when the link was established, requests for data must be outstanding or this call results in an error.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-5%)	Send Network Data Message subfunction code.
4	CHR$(*ula*%)	User link address.
5-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length of data to be sent from the buffer (bytes). If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-20	0%	Reserved -- must be zero.

Data Passed (Cont.)

Bytes	Value	Description
21	CHR$(*dmflgs*%)	Data message flags: *rls*% + *eom*% + *bom*%.

rls = 128 Link status data returned
rls = 0 No link status data returned

eom	*bom*	Meaning
0	0	Middle segment of a message
0	1	First segment of a message
2	0	Last segment of a message
2	1	Sole segment of a message

22-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.
36-40	0%	Reserved -- must be zero.
41+	*data*$	User data, if appended to the string.

Data Returned

If *rls* = 128, link status information is returned when this call is successfully completed. The data returned is identical in format to a received Link Service Message (see the section "Format of Received Link Service Message").

Discussion

ula

The user link address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user message data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. The maximum amount of data that can be transferred over this link is established by the remote system. It is passed on to the

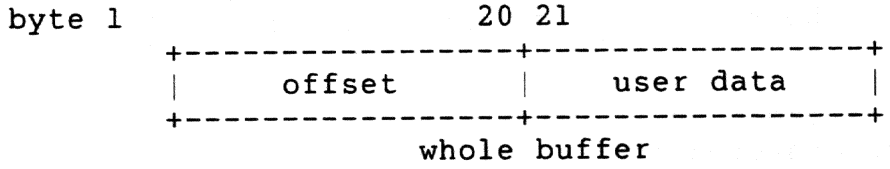
local program as a transmit maximum (the *tmax* parameter in a received Connect Initiate or Connect Confirm Message). Thus, *length* can be any value from one through *tmax*.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length is still limited by the transmit maximum.

The *length* parameter is ignored if *chnl* = 0; that is, if the user data is appended to the SYS call string.

offset

If an I/O buffer is used, the user message data can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

dmflgs

The *dmflgs* field can be coded as the sum of three bit values:

dmflgs% = *rls%* + *eom%* + *bom%*

The low-order two bits of the *dmflgs* parameter (*eom%* + *bom%*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, ECL uses these two bits to regulate the transmission of logical messages as the remote program requests them. See Chapter 4 for a full discussion of flow control.

eom	*bom*	*Meaning*
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E ECL uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is passed to the remote system and program.

ECL also uses the *eom* bit to determine whether it permits a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* bit set, ECL assumes that the last message has not been completely sent and does not permit the disconnect. This is always done, irrespective of the flow control option selected by the remote program.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, the *eom* flag should always be set. Setting the flag in this case does not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

The sign bit of this byte (*rls*) is used to request that link status information be returned to the target string when the send call has successfully completed:

rls = 128 Link status information is returned
rls = 0 No link status information is returned

If link status information is requested, it is returned in the same format as a received Link Service Message.

data

The user data to be transmitted to the remote program can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* ≠ 0). In either case, the amount of data that can be transmitted is limited by the remote program or remote ECL. This transmit maximum is passed to the calling program in the *tmax* parameter of the received Connect Initiate or Connect Confirm Message for this logical link.

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE The Data Message transmit queue for this logical link is full. ECL is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No Network Data Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a 2 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations (see the section "Buffer Allocation for Message Operations").
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be in the range of 1 to 255.

Possible Errors (Cont.)

ERR value	Meaning
19	DISK BLOCK IS INTERLOCKED Either the remote system has inhibited transmission on this link because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link). The program will be notified with a Link Service Message when the condition clears, as described in Chapter 4.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true: o The *offset* must be less than the buffer size. o The *length* must be less than or equal to the buffer size minus the *offset*. o The number of bytes to be sent as defined by *length*, or if *length* = 0, by buffer size minus *offset* must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following Send Network Data call sends user data from a buffer. The null device is opened on channel 2 to obtain buffer space, the message data is loaded into the buffer using LSET, and the user data is sent. The use of JUNK$ at the beginning of the buffer illustrates the use of the buffer offset field:

```
ULA% = 14%  !USER LINK ADDRESS = 14
DMFLGS% = 1% + 2% !SOLE SEGMENT OF LOGICAL MESSAGE
CHNL% = 2%  !I/O BUFFER ON CHANNEL 2
RIB% = 3%                !USE RIB #3
OPEN "NL:" AS FILE CHNL%, RECORDSIZE 100%
FIELD  CHNL%, 10% AS JUNK$, 90% AS TEXT$
MSG$ = "THIS MESSAGE WAS SENT FROM A BUFFER"
LENGTH% = LEN(MSG$)
OFFSET% = LEN(JUNK$)
LSET TEXT$ = MSG$
!
!EXAMPLE OF SEND NETWORK DATA MESSAGE
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-5%) + CHR$(ULA%)
      + STRING$(6%,0%) + CHR$(CHNL%) + CHR$(0%)
      + CHR$(LENGTH%) + CHR$(SWAP%(LENGTH%))
      + CHR$(OFFSET%) + CHR$(SWAP%(OFFSET%))
      + STRING$(4%,0%) + CHR$(DMFLGS%)
      + STRING$(13%,0%) + CHR$(RIB%))
```

Send Interrupt Message (DECnet)

The Send Interrupt Message SYS call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message is placed at the head of the pending message queue, behind the first message (since it can already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt Messages are subject to flow control (see Chapter 4).

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-6%)	Send Network Interrupt Message subfunction code.
4	CHR$(*ula*%)	User link address.
5-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length of data to be sent from the buffer (bytes). If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-20	0%	Reserved -- must be zero.
21	*inflgs*%	Interrupt flags. *inflgs*% = *rls*% *rls* = 128 Link status data returned. *rls* = 0 No link status data returned.
22-34	0%	Reserved -- must be zero if passed.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.

Data Passed (Cont.)

Bytes	Value	Description
36-40	0%	Reserved -- must be zero if passed.
41-56	*data$*	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

If *rls* = 128, link status information is returned when this call is successfully completed. The data returned is identical to a received Link Service Message.

Discussion

ula

The user link address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

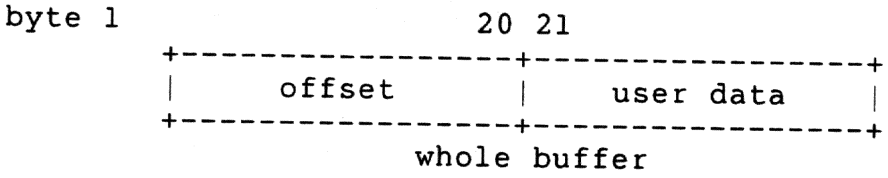
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the channel. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length is also limited to 1 to 16 bytes.

offset

If an I/O buffer is used, the user data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

inflgs

The *inflgs* parameter indicates whether link status information is to be returned to the target string when this call is successfully completed. This field should be coded as follows:

inflgs% = *rls*%

where:

rls = 128 Link status information is returned
rls = 0 No link status information is returned

If link status is requested, it is returned in the same format as a received Link Service Message.

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the I/O buffer (*chnl* ≠ 0).

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. ECL is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 2 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations (see the section "Buffer Allocation for Message Operations").
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS( ) USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified was zero. It must be within the range 1 to 255.

Possible Errors (Cont.)

ERR value	Meaning
19	DISK BLOCK IS INTERLOCKED The remote ECL has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message will be delivered to the calling program when the condition clears. See Chapter 4 for a full discussion of flow control.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true: o The *offset* must be less than the buffer size. o The *length* must be less than or equal to the buffer size minus the *offset*. o The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for an Interrupt Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
56	MISSING SPECIAL FEATURE DECnet/E was not installed at system installation time. The network functions cannot be executed.

Example

The following example shows a Send Interrupt Message call. The user data, START PHASE 2, is passed to a remote program over logical link 170 on RIB #3. Note that the remote program has to be coded to recognize this message and take appropriate action:

```
ULA% = 170%
RIB% = 3%
!
!EXAMPLE OF SEND INTERRUPT CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-6%) + CHR$(ULA%)
      + STRING$(30%,0%) + CHR$(RIB%) + STRING$(5%,0%)
      + "START PHASE 2")
```

Send Link Service Message (DECnet)

The Send Link Service Message SYS call:

- Requests data from a remote program over a flow-controlled logical link (Chapter 4)
- Reenables incoming Interrupt Messages over a logical link (Chapter 4)
- Obtains status information for the link

No user data is allowed with a Link Service Message.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-7%)	Send Link Service Message subfunction code.
4	CHR$(*ula*%)	User link address.
5-20	0%	Reserved -- must be zero.
21	CHR$(*lsflgs*%)	Link service flags. *lsflgs*% = *rls*% + *opr*%. *opr*% indicates the purpose of the call: *opr* = 0 Request data from remote program over flow-controlled link = 1 Reenable Interrupt Messages = 2 Obtain status information *rls*% indicates the whether or not link status information is returned: *rls* = 128 Link status information is returned when *opr* = 0 or 1. *rls* = 0 No link status information is returned when *opr* = 0 or 1.
22	CHR$(0%)	Reserved -- must be zero.
23	CHR$(*drcnt*%)	Data request count. Increment for logical message or counter. Must be positive and not cause counter to exceed 127. Ignored when *opr* ≠ 0.

Data Passed (Cont.)

Bytes	Value	Description
24	CHR$(*ircnt*%)	Interrupt request count. Must equal 1 when *opr* = 1. Ignored otherwise.
25-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.
36-40	0%	Reserved -- must be zero.

Data Returned

If *opr* = 2, or if *rls* = 128 when *opr* = 0 or 1, link status information is returned when this call completes successfully. The data returned is identical to a received Link Service Message.

Discussion

ula

The User Link Address (*ula*) is the link identifier (1 to 255) chosen by the calling program in its Connect Initiate Message or Connect Confirm Message sent during the connection sequence for this logical link.

lsflgs

The link service flags byte is coded as the sum of two values: *lsflgs*% = *rls*% + *opr*%. The low-order two bits (*opr*) indicate the purpose of the call:

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* parameter is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested segment or message flow control in its Connect Initiate or Connect Confirm Message for this logical link. If no flow control was requested, a Send Link Service call with *opr* = 0 fails with an error (ERR = 18).

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* parameter is interpreted as the number of Interrupt Messages allowed (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* parameters are ignored.

The sign bit of this byte (*rls*) can be used when *opr* = 0 or 1 to request that link status information be returned to the SYS call string when the call has completed successfully:

rls = 128 Link status information is returned
rls = 0 No link status information is returned

Link status information is always returned when *opr* = 2. If status information is requested, it is returned in the same format as a received Link Service Message.

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpretted as either a segment request count (segment flow control) or as a logical message request count (message flow control). In either case, it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local ECL (see Chapter 4). An error is returned if the *drcnt* given increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is meaningful only when *opr* = 1. It indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Thus, *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling incoming interrupts on the logical link. Note that interrupts can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call fails with an error (ERR=19).

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The channel specified in byte 11 is not open. The program must OPEN the channel before using it for message operations (see the section "Buffer Allocation for Message Operations").
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed causes this error.

Possible Errors (Cont.)

ERR value	Meaning
18	ILLEGAL SYS( ) USAGE One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *opr* value in byte 21 is not 0, 1, or 2. o The *opr* value is 0, but the program did not request flow control for the logical link in its Connect Initiate or Connect Confirm Message. o The data request count (*drcnt*) value given increased the count of outstanding requests to a value greater than 127. o The *opr* value is 1, indicating an Interrupt Message is being requested, but the *ircnt* argument is not equal 1. o The *opr* value is 1, but the outstanding request counter for interrupts is already equal to 1. o The *ula* given is zero. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED The *opr* value is 1, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
31	ILLEGAL BYTE COUNT FOR I/O This error can occur if byte 11 (the *chnl* number in all other system calls) is nonzero. In this case, the system attempts its consistency checking for length and offset fields in bytes 13-16 and has found them to be inconsistent. Since no message data can be transmitted with a Link Status Message, byte 11 should be zero.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.

Possible Errors (Cont.)

ERR value	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed at system installation time. The network functions cannot be executed.

Example

The following example sends a Link Service Message to request five segments from the remote program over logical link 12 on RIB #3. Assume that segment flow control was requested by the program in its Connect Initiate or Connect Confirm Message that established the logical link. For example:

```
ULA% = 12%
LSFLGS% = 0%
DRCNT% = 5%
RIB% = 3%
!
!EXAMPLE OF SEND LINK SERVICE MESSAGE CALL
!
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-7%) + CHR$(ULA%)
      +STRING$(16%,0%)+CHR$(LSFLGS%)+CHR$(0%)
      +CHR$(DRCNT%)+STRING$(10%,0%)+CHR$(RIB%))
```

Send Disconnect Message (DECnet)

The Send Disconnect Message SYS call terminates an established logical link. Messages in the pending message queue remain. However, no more messages can be sent over the link. The User Link Address (ULA) is freed and can be used for another logical link. If the last Network Data Message sent did not have the End-of-Message (EOM) flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call fails with an error.

Successful transmission of a Disconnect Message assures the sender that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not, however, guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see the section "Send/Receive Disconnect Message" in Chapter 3). Otherwise, the Disconnect Message provides no particular advantage over a Link Abort Message in terminating a logical link.

You can send up to 16 bytes of user data with the Disconnect Message.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-8%)	Send Disconnect Message subfunction code.
4	CHR$(*ula*%)	User Link Address.
5-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length of data to be sent from the buffer (bytes). If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.

Data Passed (Cont.)

Bytes	Value	Description
15-16	*offset%*	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.
36-40	0%	Reserved -- must be zero.
41-56	*data$*	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

ula

The User Link Address (*ula*) is the number (1 to 255) that the calling program established in its Connect Initiate or that the remote program sent in its Confirm Message during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, indicating that this information is appended to the SYS call string.

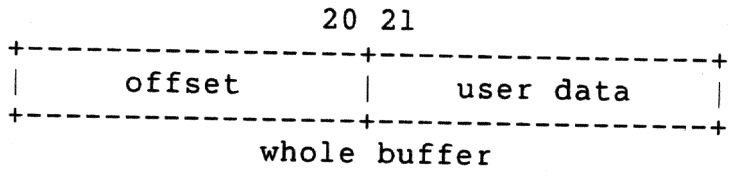
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 through 16. This field is ignored if *chnl* = 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the channel. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length must also fall in the range of 1 to 16 bytes.

offset

If an I/O buffer is used, the user data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

Up to 16 bytes of optional message data can be appended to the SYS call string (*data*) or sent from the channel buffer (*chnl* ≠ 0).

Possible Errors

ERR value	Meaning
4	NO ROOM FOR USER ON DEVICE There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, a Link Abort Message should be sent.)
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The *chnl* specified in byte 11 is not open. The program must OPEN the channel buffer before using it for message operations (see the section "Buffer Allocation for Message Operations").

Possible Errors (Cont.)

ERR value	Meaning
10	PROTECTION VIOLATION

Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed causes this error.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be within the range 1 to 255.

19 DISK BLOCK IS INTERLOCKED

The last segment sent by the local program over this logical link did not have the *eom* flag set. ECL does not permit a logical link to be disconnected unless all messages have been completely sent. Note that this test of the *eom* flag is done irregardless of the flow control option chosen by the remote program.

31 ILLEGAL BYTE COUNT FOR I/O

The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true:

- The *offset* must be less than the buffer size.
- The *length* must be less than or equal to the buffer size minus the *offset*.
- The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Disconnect Message.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store the optional message data portion of the Disconnect Message. A retry may succeed. (A request to send a Disconnect Message never fails for this reason if no user data is passed.)

Possible Errors (Cont.)

ERR value	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example disconnects logical link 230 on RIB #3 and includes user data explaining the reason for the disconnect. Note that the remote program has to be coded to recognize this message and take appropriate action:

```
!
!EXAMPLE OF SEND DISCONNECT
!
RIB% = 3%     !Set RIB number
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-8%) + CHR$(230%)
      + STRING$(30%,0%) + CHR$(RIB%) + STRING$(5%,0%)
      + "END-OF-FILE")
```

Send Link Abort Message

The Send Link Abort Message SYS call terminates a logical link immediately. Unlike the Disconnect Message, no attempt is made to make sure that all messages previously sent over this logical link have been received by the remote system. If there are still messages in the transmit queues waiting for acknowledgment from the remote ECL, the system discards them. These messages may or may not have been received by the remote system (see the section "Send/Receive Link Abort Message" in Chapter 3).

You can specify up to 16 bytes of user data in the message. If you issue the Link Abort Message for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If you issue the Link Abort Message for a link awaiting confirmation from the remote program, the user data does not reach the remote program.

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-9%)	Send Link Abort Message subfunction code.
4	CHR$(*ula*%)	User Link Address.
5-10	0%	Reserved -- must be zero.
11	CHR$(*chnl*%)	Channel number of buffer containing the user message data, 1 to 12. If 0, this data is appended to the SYS call string.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length*%	Length of data to be sent from the buffer (bytes). If 0, the number of bytes sent equals the buffer size minus *offset*. Ignored if *chnl* = 0.
15-16	*offset*%	Offset (in bytes) from the beginning of the buffer where the user data begins. If 0, the data starts at the beginning of the buffer. Ignored if *chnl* = 0.
17-34	0%	Reserved -- must be zero.
35	CHR$(R%)	Receiver ID (RIB) block number, 0 to 255.

Data Passed (Cont.)

Bytes	Value	Description
36-40	0%	Reserved -- must be zero.
41-56	*data*$	Optional user data, if appended to the string. Maximum of 16 bytes. Ignored if *chnl* ≠ 0.

Data Returned

No meaningful data is returned.

Discussion

ula

The User Link Address (*ula*) is the number (1 to 255) that the calling program established in its Connect Initiate or that the remote program sent in its Confirm Message during the connection sequence for this logical link.

chnl

The *chnl* parameter can be a number from 1 to 12, indicating which I/O buffer contains the user data to be sent to the remote program. Or it can be 0, to indicate that this information is appended to the SYS call string.

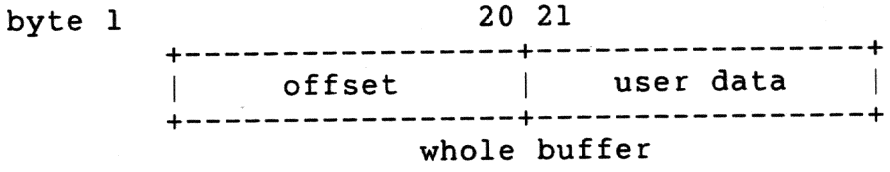
length

The *length* parameter indicates how many bytes are to be sent from the I/O buffer. Acceptable values range from 1 to 16. Or, it can be 0.

If *length* = 0, the number of bytes sent is determined from the *offset* parameter and the size of the entire buffer, as defined by the RECORDSIZE option in the OPEN statement for the buffer. That is, the number of bytes sent is equal to the buffer size minus *offset*. This length must also fall in the range of from 1 to 16 bytes.

offset

If a channel buffer is used, the user message data, if any, can be offset from the beginning of the buffer. The size of the offset (in bytes) is indicated with this parameter. For example, if *offset* = 20, the user data begins in byte 21:



The *offset* field is ignored if *chnl* = 0.

data

Up to 16 bytes of optional user data can be appended to the SYS call string (*data*) or sent from the channel buffer (*chnl* ≠ 0). This data does not reach the remote program if you issue the Link Abort Message for a link awaiting confirmation from the remote program (see the section "Send/Receive Link Abort Message" in Chapter 3).

Possible Errors

ERR value	Meaning
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in byte 4 does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
9	I/O CHANNEL NOT OPEN The *chnl* specified in byte 11 is not open. The program must OPEN the I/O buffer before using it for message operations (see the section "Buffer Allocation for Message Operations").

Possible Errors (Cont.)

ERR value	Meaning
18	ILLEGAL SYS( ) USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be in the range of 1 to 255.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are inconsistent. The following relationships must be true: o The *offset* must be less than the buffer size. o The *length* must be less than or equal to the buffer size minus the *offset*. o The number of bytes to be sent (as defined by *length*, or if *length* = 0, by buffer size minus *offset*) must be between 1 and 16 for a Link Abort Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example aborts logical link 31 on RIB #3:

```
!
!EXAMPLE OF SEND LINK ABORT MESSAGE
!
RIB% = 3%
X$ = SYS(CHR$(6%) + CHR$(22%) + CHR$(-9%) + CHR$(31%)
      + STRING$(30%,0%) + CHR$(RIB%))
```

Receive

The Receive SYS call retrieves a message from the calling program's pending message queue. It returns control parameters to the target string and user data, if any, to an I/O buffer that you specify in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the I/O buffer. The user data can be retrieved all at once or in portions, and portions can be discarded.

A Receive call can be selected to get:

- The first message in the queue
- The first message in the queue from either a network or a local sender
- The first message in the queue for a particular logical link

You can set a sleep flag and associated timer to indicate that the calling program is to wait if there are no appropriate messages pending in the queue. Otherwise, if there are no appropriate messages pending, the Receive call terminates with an error (ERR=5).

Data Passed

Bytes	Value	Description
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(2%)	Receive subfunction code.
4	CHR$(*rmod*%)	Receive modifier: *S*%+*T*%+ *L*%+*N*% *S*% Sleep/No Sleep If S%=0% and no messages are pending for this job, the receive call returns an immediate error (ERR=5). If S%=1%, the job sleeps until a message is queued. The duration of the sleep can be limited by bytes 27-28. See Discussion. *T*% Truncate/No Truncate If T%=0%, an attempt to receive a message that is too long for the buffer (indicated by bytes 11-16 of the data passed) results in a partial

Data Passed (Cont.)

Bytes	Value	Description

message being transferred to the caller. The remainder of the message is saved and can be retrieved by subsequent receive calls.

If T%=2%, a message that is too long for the buffer (specified by bytes 11-16 of the data passed) is truncated.

In either case, the number of bytes from the data portion of the message that was delivered to the buffer is returned in bytes 13-14 of the data returned. The number of bytes remaining (T%=0%) or discarded (T%=2%) is noted in bytes 9-10 of the Data Returned.

L% Local Selection

If L%=0%, local selection is disabled. If N% (described as follows) is also disabled, the first message on the receiver's queue of pending messages is delivered to the caller.

If L%=4%, local selection is enabled. Only local messages are delivered on this receive. The selection can be further qualified to a particular local sender by bytes 5 and 6.

N% Network Selection

If N%=0%, network selection is disabled. If L% is also disabled, the first message on the receiver's queue of pending messages is delivered to the caller.

If N%=8%, network selection is enabled. Only network messages are delivered on this receive. The selection can be further qualified to a particular DECnet logical link by specifying a DECnet user link address in byte 5.

Note

If L%=4% and N%=8%, the local bit setting prevails; the network selective receive is ignored.

Data Passed (Cont.)

Bytes	Value	Description
5	CHR$(*sndr*%)	Sender selection. This byte is ignored if both L%=0% and N%=0% in byte 4. Any nonzero value in this byte selects a particular sending job. Zero is a special case described for byte 6. See the Discussion for meaningful combinations of bytes 5 and 6. For local selection (L%=4% as described previously), if this byte is equal to a job number times 2, the first message on the queue from that particular job is delivered to the caller. For network selection (N%=8% and L%=0%), if this byte is equal to a user link address (ULA), the first message on the queue from that particular DECnet logical link is delivered to the caller.
6	CHR$(*qual*%)	Sender selection qualifier. This byte is ignored if both L%=0% and N%=0% in byte 4. This byte is also ignored if byte 5 is nonzero. See the Discussion for meaningful combinations of bytes 5 and 6. For local selection, if byte 5 is zero and byte 6 is nonzero, this receive is requesting a message from the "system" (represented by job 0, which is not a real job number). This special case is intended for use by ERRCPY or an EMT logger, which receives messages from the monitor's error logging or EMT logging routines. The job number in these messages is zero. See the *RSTS/E Programming Manual* for more information on EMT logging. If both byte 5 and byte 6 are zero, the selection bits (L% and N% as described above) select only the generic type of message to be delivered to the caller. For local selection (L%=4%), the first local message on the queue is delivered to the caller. For network selection

Data Passed (Cont.)

Bytes	Value	Description
		(N%=8% and L%=0%), the first network message on the queue is delivered to the caller.
7-10	0%	Reserved -- must be zero.
11	CHR$(*chnl%*)	Channel number for the I/O buffer to receive messages. If *chnl%* is between 1 and 12, the system returns the data portion of the message in the buffer for channel *chnl%*. The channel must be open. If *chnl%*=0 or the buffer for *chnl%* is not large enough to accommodate the data portion of the message, the action taken depends on the value of the truncation bit in the receive modifier. See Discussion. Channel 0 can be used for the I/O buffer if 128 is added to the channel number, that is, CHR$(128%+0%). In general, CHR$(128%+*chnl%*) allows channels 0 through 12 to be used for I/O buffers.
12	CHR$(0%)	Reserved -- must be zero.
13-14	*length%*	Maximum message length (in bytes) desired on this receive in the form CVT%$(SWAP%(*length%*)). If byte 11 is zero (that is, no channel is specified), the offset and length fields are ignored. The length field limits the number of data bytes that are returned on this receive. If the length is zero, the error ?No room for user on device (ERR=4) occurs. Otherwise, a maximum of *length%* bytes is returned on this receive. The specified length must be less than or equal to the buffer size minus the specified offset.

Data Passed (Cont.)

Bytes	Value	Description
15-16	*offset*%	Offset from the start of the buffer in the form CVT%$(SWAP%(*offset*%)). The offset field determines where in the buffer the data portion of the message is returned. The offset value is added to the location of the beginning of the buffer. The offset value must be in the range 0 to (size of buffer -#1).
17-26	CHR$(0%)	Reserved -- must be zero.
27-28	*slptime*%	Sleep time in seconds in the form CVT%$(SWAP%(*slptime*%)). If byte 4 requests a sleep and no messages are pending, the sleep is terminated after *slptime*% seconds. If *slptime*%=0%, the length of the sleep is indefinite; the job is not awakened until one of six events awakens the job. When the sleep terminates, the error ?Can't find file or account (ERR=5) occurs. See Discussion.
29-34	CHR$(0%)	Reserved -- must be zero.
35	CHR$(*rib*%)	Receiver ID (RIB) block number for this receive. The RIB number must be a value from 0 to 255; values 128 through 255 are reserved for use by DIGITAL.
34-40	CHR$(0%)	Reserved -- must be zero.

Discussion

On any receive call, the system checks the eligibility of the job to receive messages and returns the error ?Illegal SYS() usage (ERR=18) if the specified job and RIB number are not in the list of declared receivers. If the job is eligible to receive messages, the call attempts to receive a message based on the receive modifier passed in byte 4. Normally, a receive call returns the first message on the receiver's queue of pending messages. Use the selective receive bits (L% and N% in byte 4) to select messages from particular senders identified by the local job number or DECnet user link address (as indicated by byte 5). If the sleep bit is off (S%=0% in byte 4) and no messages are pending, the system generates the error ?Can't find file or account (ERR=5) and immediately passes the error to the calling program. If no messages are pending and the sleep bit is on

(S%=1% in byte 4), the job is put into a sleep state (called a receiver sleep). You can specify the sleep time in bytes 27-28 of the data passed.

A job in a receiver sleep can be awakened by any of six events:

- A user types a delimiter (RETURN, LINE FEED, FORM FEED, or ESCAPE) at:
  - Any terminal opened by the job.
  - Any terminal allocated to the job if the job also has a keyboard open on a nonzero channel.
- A dial-up line that is allocated or opened by the job gets hung up.
- The system manager disables logins (that is, sets the number of logins to 1).
- A state change occurs on a pseudo keyboard opened by the job. This condition can occur when the opened pseudo keyboard has output for the controlling job or has entered an input wait state.
- The job has declared itself a receiver and a message is queued for it through the Send/Receive SYS calls.
- The job has a DMC/DMR (XM:) device open and the device driver receives a message.

In all cases, the job is awakened with an error ?Can't find file or account (ERR=5) but is not passed a message. To obtain a pending message, the job must execute the receive call again. Because the job may have been awakened by terminal input or expiration of the timer, you can check for pending messages by executing the receive call without a sleep or by executing a terminal input operation using RECORD 8192% for immediate return.

The receive call returns parameters in the target string and the data portion of the message (if any) in the I/O buffer specified by byte 11. If the program must handle any DECnet/E messages, or local messages longer than the 20 bytes of user-defined parameters, a channel buffer must be available to receive the data portion of the message.

You can determine the number of bytes from the data portion of a message actually delivered to the buffer (if any) and the number of bytes remaining in the message (if any) from the data returned with

the receive call. Bytes 13-14 indicate the number of bytes from the data portion of the message that were delivered to the buffer. The truncation bit (T%) in byte 4 determines whether the remaining bytes in the channel buffer are kept or discarded. If you set T%=0%, the remaining bytes are kept. If T%=2%, the remaining bytes are discarded. Bytes 9-10 indicate the number of bytes that remain to be transferred or were discarded (depending on the truncation bit in byte 4).

When processing large messages in small pieces, each successive receive call retrieves a limited number of bytes from the same message. The normal sequence is to issue receive calls until the number of bytes remaining in the data portion of the message is zero (as indicated by bytes 9-10 of the data returned). The receiver then knows that the entire message has been delivered and removed from the queue.

A convenient way to assign a buffer for message operations is to open the null device (NL:) at the desired buffer size with the RECORDSIZE option in the OPEN statement. The null device is always available and can be opened as many times as required to obtain buffer space for any desired function. If you specify a buffer, the system ensures that the channel is open. If the channel is not open, the call results in an immediate error ?I/O channel not open (ERR=9).

The program receiving a message selects the particular sender by combining receive modifier bits (L% and N% in byte 4) and the values of bytes 5 and 6.

Table 5-4 summarizes the possible combinations.

Table 5-4: Sender Selection Summary

Data Passed				
Receive Modifier				Result
Byte 4 N%	Byte 4 L%	Byte 5	Byte 6	
0%	0%	--	--	Bytes 5 and 6 ignored; returns first queued message.
-	4%	0%	0%	Selects first local message.
		0%	nonzero	Selects job 0; used by error logging and EMT logging programs to select messages from monitor routines.
		nonzero	--	Selects local message by job number times 2 in byte 5.
8%	0%	0%	0%	Selects first network message.
		nonzero	--	Selects network message by link (user link address) in byte 5.

Possible Errors

ERR value — **Meaning**

5 CAN'T FIND FILE OR ACCOUNT

If a receive without sleep was issued, this error indicates that no messages are pending. If a receive with sleep was issued, this error indicates that no messages were pending when the receive call was issued or that the sleep timer has expired. The error is returned withn the job is awakened from the sleep. The program must execute the receive again to retrieve any pending messages.

9 I/O CHANNEL NOT OPEN

An attempt was made to receive a message, but the channel specified in byte 11 of the data passed is not open. The program must open the channel and try again.

Possible Errors (Cont.)

ERR value	Meaning
18	ILLEGAL SYS( ) USAGE The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
31	ILLEGAL BYTE COUNT FOR I/O The *offset* and/or *length* fields passed in bytes 13-16 are invalid. The following relationships must be true: o The *offset* must be less than the buffer size. o The *length* must be less than or equal to the buffer size minus *offset*. The *offset* and *length* fields are checked for validity whenever a channel number is passed in byte 11.

Format of Received Local Data Message

The following format describes the information returned to the target string for a Local Data Message. Up to 512 bytes of user data can be delivered to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-1%)	Local Data Message subfunction code.
4	CHR$(*jobx2%*)	The job number of the local sender times two.
5-6	*ppn%*	Project-programmer number of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	*remainder%*	Number of bytes of user data not delivered to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set.
11-12	--	Reserved.
13-14	*length%*	Length (in bytes) of information transferred to the I/O buffer on this Receive call.
15-20	--	Reserved.
21-40	*param$*	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Format of Received Local Data Message with Privilege Mask

The following format describes the information returned to the target string for a Local Data Message with Privilege Mask.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-11%)	Local Data Message with Privilege Mask subfunction code.
4	CHR$(*jobx2%*)	The job number of the local sender times two.
5-6	*ppn%*	Project-programmer number of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	*remainder%*	Number of bytes of user data not delivered to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set.
11-12	--	Reserved.
13-14	*length%*	Length (in bytes) of information transferred to the I/O buffer on this Receive call.
15-20	--	Reserved.
21-28	*priv$*	The sender's privilege mask.
21-40	*param$*	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Format of Received Connect Initiate Message

The following format describes the information returned to the target string for a Connect Initiate Message. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-2%)	Connect Initiate Message subfunction code.
4	--	Reserved.
5-6	*lla%*	Local link address. Identifies the link for the local program's subsequent Connect Confirm or Connect Reject Message.
7-8	--	Reserved.
9-10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set.
11-12	--	Reserved.
13-14	*length%*	Length (in bytes) of data transferred to the I/O buffer on this Receive call.
15-22	--	Reserved.
23	CHR$(*rlmod%*)	Remote link modifier. Indicates the flow control in effect for the remote program. The two low-order two bits should be masked and interpreted as follows: *flow%* = *rlmod%* AND 3% If *flow* = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.

Format of Received Connect Initiate Message (Cont.)

Bytes	Value	Description
25-26	*rmax*%	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. The *rmax* value is supplied by the local ECL and is calculated from the size of the receive buffers it allocates. The maximum size of the receive buffers is set for ECL by Network Management. The local program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a connection (see the section "Send Connect Confirm Message").
27-28	*tmax*%	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Figure 5-2 shows the format of a received Connect Data Block. Table 5-5 describes the format in detail.

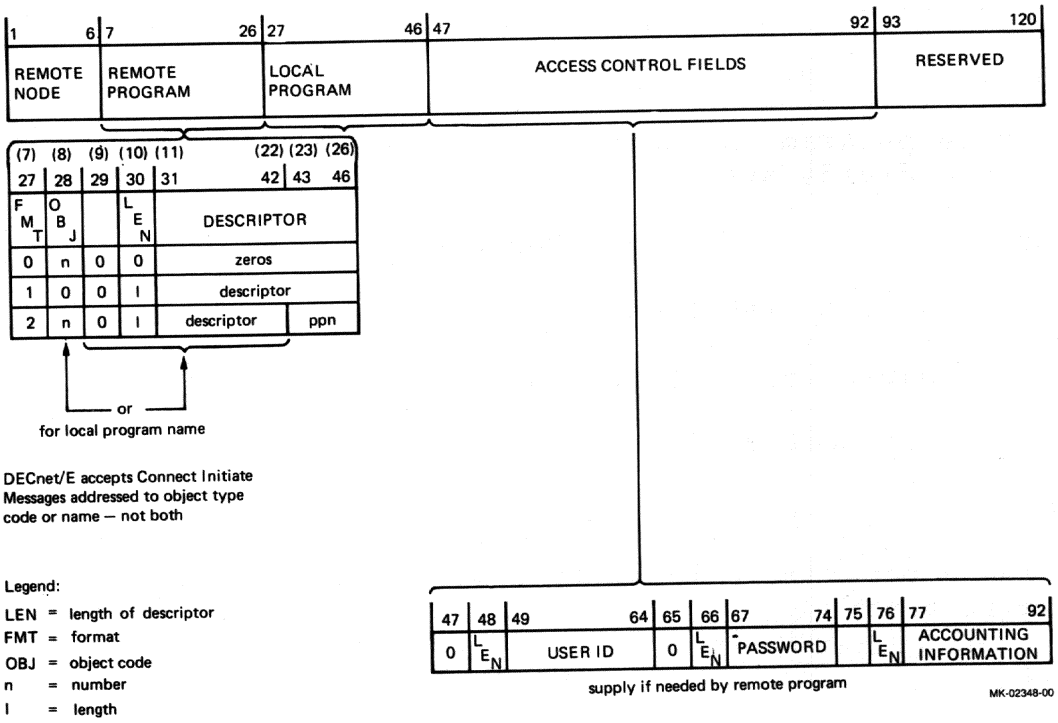


Figure 5-2: Format of Received Connect Data Block

Table 5-5: Format of Received Connect Data Block

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node from which the Connect Initiate Message was received. Names shorter than 6 characters are left-justified with space fill to 6 characters.
7-26	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of the following three formats.

Table 5-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 0 The remote program is identified by object type code alone. The fields are: **Bytes** — **Content** 7 — Zero, to indicate a format 0 name. 8 — Object type code of the remote program. 9-26 — Zero (not used).
	Format 1 The remote program is identified by a descriptor. The subfields are: **Bytes** — **Content** 7 — One, to indicate a format 1. 8 — Normally zero. This byte is usually zero unless the target DECnet system identifies the remote program with both name and object type code on outgoing connect requests. 9 — Reserved. 10 — Length (in bytes) of the descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26). 11-26 — Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Table 5-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 2 This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC. DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved.
10	Length (in bytes) of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor is left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E PPN.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E PPN.

Table 5-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
27-46	**Local Program Name.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Equal to 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.

Table 5-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
47-92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most precise checking done in a normal login on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system passes this information on to the receiving program. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49-64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN at login.
65	Reserved.
66	Length of password information specified in the next field, 0 to 8 bytes.
67-74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login.

Table 5-5: Format of Received Connect Data Block (Cont.)

Bytes	Content	
	75	Reserved.
	76	Length of the accounting information specified in the following field, 0 to 16 bytes.
	77-92	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.
93-120	Reserved.	

Format of Received Connect Confirm Message

A Received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the receiving program. The link is identified by the User Link Address (ULA) specified by the local program in the Connect Initiate Message. The following format describes the information returned to the target string. You can also return up to 16 bytes of user data to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-3%)	Connect Confirm Message subfunction code.
4	CHR$(*ula*%)	User Link Address.
5-8	--	Reserved.
9-10	*remainder*%	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the *t* flag was set in the data passed.
11-12	--	Reserved.
13-14	*length*%	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15-22	--	Reserved.
23	CHR$(*rlmod*%)	Remote link modifier. Indicates the flow control in effect for the remote program. The two low-order bits should be masked and interpreted as follows: *flow*% = *rlmod*% AND 3% If *flow* = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.

Format of Received Connect Confirm Message (Cont.)

Bytes	Value	Description
25-26	*rmax*%	Receive maximum, in bytes. The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This is: o Equal to the *rmax* value specified in the Connect Initiate Message for this link. o A lesser value supplied by the local ECL, calculated from the size of the buffers that it allocates for receives.
27-28	*tmax*%	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Format of Received Connect Reject Message

A Received Connect Reject Message is a rejection of the receiving program's request for a logical link. It can be from the remote program or from the local or remote ECL. The link is identified by the User Link Address (ULA) that the local program established in its Connect Initiate Message. The following format describes the information returned to the target string. Rejections from ECL contain a reason code explaining the reason for the rejection and do not contain user data. Rejections from the remote program can contain up to 16 bytes of user data delivered to the I/O buffer.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-4%)	Connect Reject Message subfunction code.
4	CHR$(*ula*%)	User Link Address, previously defined in the local program's Connect Initiate Message.
5-8	--	Reserved.
9-10	*remainder*%	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the t flag was set in the data passed.
11-12	--	Reserved.
13-14	*length*%	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15-22	--	Reserved.
23-24	*reason*%	Identifies the reason for the rejection (see Appendix B).
25-40	--	Reserved.

Format of Received Network Data Message

A Received Network Data Message contains user data from a remote program. The following format describes the information returned to the target string. The user data accompanying the Network Data Message is delivered to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-5%)	Network Data Message subfunction code.
4	CHR$(*ula%*)	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder%*	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the t flag was set in the data passed.
11-12	--	Reserved.
13-14	*length%*	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15-20	--	Reserved.

Format of Received Network Data Message (Cont.)

Bytes	Value	Description
21	CHR$(*dmflgs%*)	Data message flags. The two low-order bits indicate whether this is the beginning, middle, end, or sole segment of a logical message. These bits should be masked and interpreted as follows: *bom%* = *dmflgs%* AND 1% *eom%* = *dmflgs%* AND 2% Where: *eom* *bom* Meaning 0 0 Middle segment of message 0 1 First segment of message 2 0 Last segment of message 2 1 Sole segment of message DECnet/E uses the *eom* bit to determine the end of a logical message if the local program requested message flow control when the logical link was established. It does not use the *bom* bit but passes it on to the receiving program.
22-40	--	Reserved.

Format of Received Interrupt Message

The local ECL places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The following format describes the data returned to the target string. You can deliver up to 16 bytes of user data to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-6%)	Network Interrupt Message subfunction code.
4	CHR$(*ula*%)	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder*%	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the t flag was set in the data passed.
11-12	--	Reserved.
13-14	*length*%	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15-40	--	Reserved.

Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link and is only returned on a Receive call when the pending message queue has emptied after a condition has cleared that previously inhibited the local program from sending (see the section "Flow Control" in Chapter 4). Status information with the same format as a Link Service Message can also be returned to the target string on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The following format describes the information returned to the target string. No data is delivered to the I/O buffer with a Link Service Message.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-7%)	Link Service Message subfunction code.
4	CHR$(*ula*%)	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	0%	Always zero for Link Service. (These bytes are the *remainder* field in other returned messages.)
11-12	--	Reserved.
13-14	0%	Always zero for Link Service. (These bytes are the *length* field in other returned messages.)
15-20	--	Reserved.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
21	CHR$(*lstat%*)	Local status flags. These flags should be masked and interpreted as follows: *data%* = *lstat%* AND 1% *intls%* = *lstat%* AND 2% *lclbp%* = *lstat%* AND 192% (2-bit field) If *data* = 1, outgoing Network Data Messages are inhibited. (A Link Service Message is delivered when the condition clears.) If *intls* = 2, outgoing Interrupt and Link Service Messages are inhibited. (A Link Service Message is delivered when the condition clears.) The *lclbp* value indicates the local ECL's backpressure status: 0 Incoming Data Message flow is on. 64 Incoming Data Message flow is on but the local ECL turns this link off if another Data Message is received before the local program's receive queue is emptied. 128 Incoming Data Message flow has been turned off by the local ECL. The local program's receive queue must be emptied before the link is turned on. 192 Incoming Data Message flow has been turned off by the local ECL but flow is scheduled to turn on as soon as system buffers become available.
22	CHR$(*rstat%*)	Remote status flags. Only one bit is currently defined for this field. This bit should be masked and interpreted as follows: *rembp%* = *rstat%* AND 128% If *rembp* = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
23	CHR$(*ldr%*)	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	CHR$(*lir%*)	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of interrupts requested but not yet received. This count never exceeds one.
25	CHR$(*rdr%*)	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	CHR$(*rir%*)	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote program, reflecting the actual count of Interrupt Messages requested but not yet sent.
27	CHR$(*dtm%*)	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant that is set when the system manager enables DECnet/E. It applies individually to all active logical links.
28	CHR$(*dtc%*)	Data transmit queue count. The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote ECL.
29	CHR$(*ltm%*)	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant that is set when the system manager enables DECnet/E. It applies individually to all active logical links.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
30	CHR$(*ltc*%)	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote ECL.
31	CHR$(*mmax*%)	Message maximum. The maximum number of incoming messages that will be queued for the local program. This maximum is set by the program in the Declare Receiver call.
32	CHR$(*mcnt*%)	Message count. The total number of messages currently in the pending message queue for the local program.
33	CHR$(*mula*%)	Message count for this logical link. The number of messages currently in the pending message queue for this logical link, as identified by the *ula* (byte 4).
34-40	--	Reserved.

Format of Received Disconnect Message

A Received Disconnect Message indicates that an established logical link with a remote program has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The following format describes the data returned to the target string. You can return up to 16 bytes of user message data to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-8%)	Disconnect Message subfunction code.
4	CHR$(*ula*%)	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder*%	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the t flag was set in the data passed.
11-12	--	Reserved.
13-14	*length*%	Length (in bytes) of data that was returned to the I/O buffer on this Receive.
15-40	--	Reserved.

Format of Received Link Abort Message

A Received Link Abort Message indicates that a logical link has been terminated by the remote program or by the local or remote ECL. The following format describes the data returned to the target string. You can return up to 16 bytes of user data to the I/O buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	CHR$(-9%)	Link Abort Message subfunction code.
4	CHR$(*ula*%)	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-6	*lla*%	Local link address, established by the local ECL for this link. It identifies the link if the remote program has sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists to identify the link being aborted under these conditions.
7-8	--	Reserved.
9-10	*remainder*%	Number of bytes of data not returned to the I/O buffer. This data has been either discarded or saved for a later Receive, depending on how the *t* flag was set in the data passed.
11-12	--	Reserved.
13-14	*length*%	Length (in bytes) of data that was returned to the I/O buffer on this Receive call.
15-22	--	Reserved.

Format of Received Link Abort Message (Cont.)

Bytes	Value	Description
23-24	*reason*%	Abort Reason (interpreted as a 16-bit integer). If this value is nonzero, the link was aborted by either the local or remote ECL and the reason is specified in these two bytes. See Appendix B for a list of reasons that apply for a link abort. If *reason* is zero, the link was aborted at the request of the remote program.
25-40	--	Reserved.

Chapter 6

Network Programming in BASIC-PLUS-2

This chapter presents the specific calls, with detailed formats, used to access the DECnet/E message services in the BASIC-PLUS-2 programming language.

Programming Background

To use the DECnet/E message services in BASIC-PLUS-2, you code subroutine calls with the CALL BY REF statement. The calls reference external MACRO subroutines stored in the system library that must be linked with your program. Thus, the process involves coding, compiling, and linking. The following sections briefly describe these aspects of using DECnet/E in BASIC-PLUS-2. Specific call formats are given in the sections "Declare Receiver (MDCL)" through "Receive (MRCV)."

Coding

The general format of the RSTS/E message calls in BASIC-PLUS-2 is:

```
CALL name BY REF(arg1, arg2, ...)
```

where:

name is the name of one of the 14 subroutines available for message services. Each subroutine processes a call at execution time by executing the following steps:

1. Checking for the proper number of arguments in the call.
2. Translating the call and arguments into the form recognizable to the RSTS/E Monitor.

3. Passing control to the RSTS/E Monitor which in turn invokes the DECnet/E software.

DECnet/E processes the request and returns control to the user program. If an error occurs, DECnet/E returns an error value to an integer variable defined as an argument in each call.

arg... is the argument list for the particular call. Values passed can be any valid BASIC-PLUS-2 expression of the appropriate type. For example, if the description of an argument calls for an integer value, you can supply any valid BASIC-PLUS-2 integer constant, variable, or expression.

Note

CALLs do not do implicit type conversions. That is, if an integer value is required and you specify a floating point value, the floating point value is passed to the message subroutine. Unexpected fatal errors can result.

Most calls contain arguments to define buffer areas for sending and receiving data. A discussion follows on how buffers can be set up for Send/Receive calls.

Setting Up Buffers

You specify either two or three arguments in the DECnet/E message calls to define a buffer: a starting address, a length value defining the number of bytes in the buffer, and in some cases, an offset in bytes from the beginning of the buffer. In all cases, the Send/Receive calls use a contiguous area in computer memory to send from or receive to. Thus, for most applications, it is better to preallocate space for send and receive buffers rather than to use dynamic strings or arrays. The following discussion addresses both methods.

Preallocating Space

Use the DIM, COMMON, or MAP statements to allocate a contiguous block of memory at compile time for send and receive buffers. The MAP statement is particularly useful. It allocates space and allows the definition of fields within the buffer having different data types. For example, you can define a map with a string field, followed by

numeric values, followed by another string. The same area can also be remapped with different field definitions. For example:

```
MAP (RETBUF) A$=40%
MAP (RETBUF) FILL%,CODEULA%,LLA%,FILL%,REMAINDER%, &
             FILL%,LENGTH%,FILL$=6%,PARAM$=20%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,LLA%,FILL%,REMAINDER%, &
             FILL%,LENGTH%,FILL$=6%,PARAM$=20%
```

The first MAP statement allocates a 40-byte area. The MAP name is RETBUF and A$ can serve as the starting address argument for a message call.

The next two MAP statements show two ways of getting at integer byte values in BASIC-PLUS-2. The second MAP statement shows an integer field CODEULA%, a one-word field that is to be referenced as a one-byte code followed by a one-byte user link address. Once data is in the buffer, you can reference the code byte with the following statement:

```
CODE%=SWAP(CODEULA%) AND 255%
```

You can reference the *ula* byte with a statement:

```
ULA%=CODEULA% AND 255%
```

The third MAP statement shows the same area in the buffer as two one-byte string variables, CODE$ and ULA$. You can then refer to bytes three and four as integer values with the following statements:

```
CODE%=ASCII(CODE$)
ULA%=ASCII(ULA$)
```

Using Dynamic Strings and Arrays

A dynamic string or array is any string or array for which space has not been preallocated with a DIM, COMMON, or MAP statement. BASIC-PLUS-2 allocates space for dynamic strings and arrays at execution time, not at compile time. Except for dynamic string arrays, you can use dynamic values for send requests, as shown in the following section, "Buffer Arguments." The elements of dynamic string arrays are not stored contiguously in memory. Thus, a send request referring to an entire dynamic string array could include data that is not part of the array itself.

Do not use dynamic strings and arrays for receive requests. The amount of data stored in a buffer by a receive request is limited only by the length argument in the Receive call. It is not limited by the length of the dynamic string or array. Thus, if the data returned by the Receive call is longer than the space currently allocated for the dynamic string or array, the data stored by the receive destroys some

quantity of data in locations that are uncontrolled by the user program.

Buffer Arguments

Starting Address: For buffers from which data is to be sent, you can specify any valid BASIC-PLUS-2 constant or variable for this argument, except a variable naming an entire dynamic string array. For example, if you want to send pure numeric data, you could specify this argument as an array name in the format A(), B%(,), and so forth.

Note

> BASIC-PLUS-2 stores two-dimensional array values in contiguous locations by row -- (0,0) first, (0,1) second, (0,2) third, and so forth. This can be significant if the data is received by a remote FORTRAN program that assumes two-dimensional arrays are stored by column.

It can also be practical to specify a string constant as the Starting Address argument for certain types of Send calls. For example, suppose you want to notify the remote program of a reason for aborting a logical link. You can code the call with the string constant ENDFILE as the starting address argument. However, you must make sure that the length argument is correct.

For buffers that are to receive data, the starting address argument should name some variable for which space has been preallocated with DIM, COM, or MAP, as discussed previously.

Offset: This argument can be any valid integer value. It defines an offset (in bytes) from the starting address where the data to be sent begins. This is useful if you are sending segments from a buffer containing an entire logical message.

Length: This argument can be any valid integer value. It defines the total number of bytes to be sent or the maximum number of bytes to be received. Each character of a string is 1 byte, an integer value is 2 bytes, a single-precision floating point value is 4 bytes, and a double-precision floating point value is 8 bytes.

Declare Receiver: MDCL (Local and DECnet)

You must execute the MDCL call before a program can receive any messages or send any network messages. (Local messages can be sent without first executing this call.) The call defines an identifying object type or name, or both, and any restrictions on incoming messages. The Monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block (RIB) for the job (see the section "DECnet/E Use of the Declare Receiver Call" in Chapter 3). It remains in effect until a MREM (Remove Receiver) call is executed for the job.

BASIC-PLUS-2 Call Format

```
CALL MDCL BY REF(errval%,name$,objtyp%,access%,0%,
                 bmax%,mmax%,lmax%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MDCL call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

name

The logical name is a string containing 1 to 6 characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name is left-justified in the string and (if necessary) padded to six characters with spaces. (A store into a 6-byte string defined in COMMON or MAP does this automatically.) Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), the string should contain six spaces.

Local programs can address the calling program by the logical name. Network programs can use either the logical name or the network object type code, if one is declared.

If the logical name is null, only network access by object type is allowed. (See the following discussion of *name*, *objtyp*, and *access*.) Otherwise, the logical name must be nonnull and unique.

objtyp

This integer value specifies the network object type code, another form of network addressing. As discussed in section "Declaring Identity" in Chapter 3, the object type code defines some service the program performs. If the program issuing the MDCL call is addressed

by object code alone (*name* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environment. Unlike the logical name, the object type code need not be unique within the node -- multiple copies of a program can declare their identity with the same type code at the same time. If *name* is null, *objtyp* cannot be zero. See Chapter 4 for a detailed discussion on network addressing.

Acceptable values for the network object type code range from 0 through 255. The value 0 indicates that object code addressing is not being used for the calling program. The range from 1 to 127 is reserved for DECnet use (see Appendix A). The range from 128 to 255 is available for definition and use by a network installation.

access

This integer value consists of the sum of five bit values (*l*% + *p*% + *n*% + *o*% + *s*%) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values (*l*%, *p*% and *n*%) limit the types of senders that can queue messages for the calling program. However, they do not limit the messages that the program can send.

If *l* = 0, messages from local senders are not queued for the calling program. (Local senders who use the network functions are considered network senders in this context.) If *l* = 1, messages from local senders are queued.

If *p* = 0, incoming messages from both local senders with privilege and without privilege are queued for the calling program. If *p* = 2, incoming messages from local senders are queued only if the sender has SEND privilege. (This bit is ignored if *l* = 0, that is, if no local senders are allowed.)

If *n* = 0, Session Control rejects incoming requests for logical links If *n* = 4, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared message maximum (*mmax*) and the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access.

Table 6-1 summarizes access code values and the access permitted for each.

Table 6-1: Types of Receiver Access

Access Code (1%+p%+n%)	Network Logical Links	Local Senders with Privilege	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

The bit value *o* is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If *o* = 0, incoming requests for logical links do not affect the program's link maximum. However, if *o* = 8 after queuing an incoming Connect Initiate Message to the receiver, the End Communication Layer (ECL) sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 8, a program prevents the possibility of a Connect Initiate Message being queued to it after its one logical link is disconnected and before it issues a Remove Receiver call. For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote system, Session Control queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that the first link is broken and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value *s* is used to modify the function of the RSTS/E conditional Sleep monitor call. If *s* = 0, any unreceived message queued for the program blocks the execution of a conditional Sleep call. This is the normal RSTS/E function. However, if *s* = 16 the RSTS/E monitor does

not check the program's message queue when determining whether it should suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can block the Sleep call but a pending message does not.

name, *objtyp*, and *access*

Session Control checks the *name*, *objtyp*, and *access* arguments to make sure that they are consistent. For example, a null logical name and an access allowing local senders are inconsistent since local senders can address the program by logical name. The system validates these arguments as shown in Table 6-2. If the arguments are invalid, the system returns an error code of 18 in the *errval* argument. If the arguments are valid, the system allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (*errval* = 4). A retry may succeed.

Table 6-2: System Validation of Name, Object, and Access Parameters

Access Code (l%+p%+n%)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local (only)	Ignored	Must be nonnull and unique.
4,6	Network (only)	Zero	Must be nonnull and unique.
		Nonzero	If *name* is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If *name* is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.

Table 6-2: System Validation of Name, Object, and Access Parameters (Cont.)

Access Code (l%+p%+n%)	Senders Permitted	Object Type	Logical Name
5,7	Network or Local	Any value	Must be nonnull and unique so local senders can refer to the calling program by logical name. Network senders can use the logical name or object type (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

mmax

The message maximum (*mmax*), an integer value ranging from 0 to 255, limits the number of messages that are queued for the calling program.

ECL keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the message maximum set by the receiver:

- Incoming Local Data Messages are not queued for the receiving program. An error is returned to the local sender.
- Incoming Connect Initiate Messages cause ECL to determine if another copy of the program can be started automatically (see Chapter 4, "Handling of Incoming Connect Requests"). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
- An incoming Network Data Message causes ECL to inhibit incoming data messages on that particular logical link. This forces the remote system to retransmit when flow is reenabled. See Chapter 4, "Backpressure Flow Control," for further details.

lmax

The link maximum (*lmax*), an integer value, limits the number of incoming requests for logical links for the calling program. If the number of links currently active is greater than or equal to *lmax* (0 to 63) when a remote Connect Initiate Message is received for the

program, the local Session Control determines if another copy of the program can be started automatically (see Chapter 4, "Handling of Incoming Connect Requests"). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting *n* = 0 in the access argument. That is, Session Control rejects all incoming requests for logical links to the program. The *lmax* argument does not limit the number of logical links initiated by the program. That is, it does not stop the program from sending Connect Initiate Messages.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs), numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call. The system allocates space for each RIB from the small buffer pool.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE

The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the last case, issue a Remove Receiver call followed by another Declare Receiver call. (It is common practice to code a Remove Receiver call immediately before a Declare Receiver call.)

4	NO ROOM FOR USER ON DEVICE

There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.

10	PROTECTION VIOLATION

The calling program must have SYSIO privilege at the time the Declare Receiver call is executed.

16	NAME OR ACCOUNT NOW EXISTS

The logical name specified is being used by another job.

18	ILLEGAL SYS() USAGE

The parameters specified are inconsistent. See the discussion of *logical name*, *object*, and *access*.

Example

The following example shows a receiver declaration allowing both network and local access. Both a logical name (BUS) and an object type code (142) are declared. Note that 6 bytes for the logical name were preallocated with COM. Thus, when the three characters BUS are stored in LNAME$, they are stored left-justified with space fill.

The buffer maximum is set to 512 and the message maximum is set to 5. No more than 512 bytes of user message data from incoming Local Data Messages are stored in Monitor pool space. And, if more than five messages accumulate for the program, an incoming request for a connection causes ECL to check to see if another copy of the program can be automatically started. If not, ECL rejects the link and any further incoming data messages causes ECL to inhibit incoming message flow on all the program's logical links because of a backpressure

condition. The link maximum is set to 2. When the program has established two logical links, an incoming Connect Initiate Message causes NSP to check to see if another copy of the program can be started. If not, NSP rejects the link. The RIB # is 17. For example:

```
1000    COM LNAME$ = 6% &
        \ERRVAL% = 0% &
        \LNAME$ = "BUS" &
        \OBJTYP% = 142% &
        \ACKSES% = 5% &
        \BMAX% = 512% &
        \MMAX% = 5% &
        \LMAX% = 2% &
        \RIB% = 17%
           .
           .
           .
2000    CALL MDCL BY REF(ERRVAL%,LNAME$,OBJTYP%,ACKSES%, &
             0%,MMAX%,LMAX%,RIB%) &
        \IF ERRVAL%_<>0% THEN GOTO 12000
           .
           .
           .
```

Remove Receiver: MREM (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. Session Control discards all pending messages and aborts any active logical links. Execute this call at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Programs with JOBCTL privilege can remove other RSTS/E jobs with this call although normally, only system utility programs would do this.

BASIC-PLUS-2 Call Format

```
CALL MREM BY REF(errval%,jobx2%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MREM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

jobx2

A zero value for *jobx2* removes the calling job. Programs with JOBCTL privilege can remove another local RSTS/E job by specifying job number times two for this integer argument.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
10	PROTECTION VIOLATION The caller does not have JOBCTL privilege and attempted to remove another job (that is, *jobx2* is nonzero).
18	ILLEGAL SYS() USAGE The *jobx2* argument was odd. *Jobx2* must be zero to remove the calling program or job number times two to remove another job.

Example

The following example shows a typical Remove Receiver request removing the calling job's RIB #17 from the system's list of declared receivers:

```
\ERRVAL% = 0% &
\RIB% = 17% &
\CALL MREM BY REF(ERRVAL%,0%,RIB%)
```

Get Local Node Parameters: NTLN (DECnet)

The NTLN call returns information to the calling program concerning the local node's network parameters. Data returned includes the node name or node number, address, and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. See the section "Automatic Job Start-Up" in Chapter 4 for a discussion of the default user's account.

BASIC-PLUS-2 Call Format

```
CALL NTLN BY REF(errval%,buffer$,buflen%,bufoff%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

buffer

This argument defines the starting address of the buffer to receive the local node information. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *buffer*.

When the NTLN call completes successfully, data is returned to the buffer in the following format:

Bytes	Contents
0-5	Local node name, consisting of from 1 to 6 uppercase, alphanumeric ASCII characters. The name contains at least one alphabetic character. A name shorter than 6 characters is left-justified and padded to 6 characters with spaces.
6-13	Reserved.
14-15	Local default PPN. Byte 14 = programmer number; byte 15 = project number. If no default account is defined, both bytes are zero.

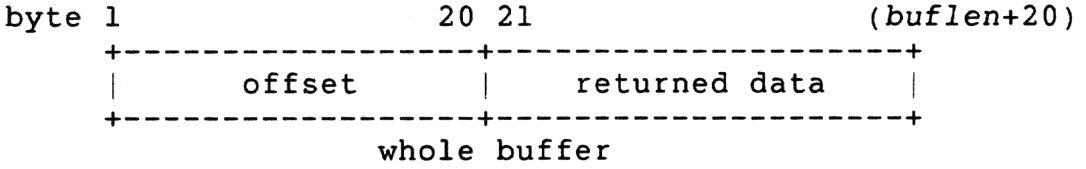
buflen

This integer argument defines the number of bytes to be returned to the buffer specified with *buffer*. If *buflen* is less than 16 bytes, the call returns only as much data as there is room for. For example,

if *buflen* = 8 bytes, the call returns only the local node name and address. Sixteen bytes are required to receive all the local node parameters.

bufoff

This integer value defines an offset, in bytes, from the address specified by the *buffer* argument. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if offset = 20, the returned data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of 10. If the actual buffer, beginning at *buffer* was only 16 bytes long, the returned data would overwrite 10 bytes beyond the end of the buffer.

Possible Errors

errval	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTLN call used to obtain all the local node parameters. The node name is to be stored in the string variable NAME and the default PPN, in the integer variable PRJPRG:

```
100       MAP (RETBUF) NODDAT$ = 20% &
          \ MAP (RETBUF) NAME$=6%,FILL$=8%,PRJPRG%
            .
            .
            .
200       ERRVAL% = 0% &
          \ CALL NTLN BY REF(ERRVAL%,NODDAT$,LEN(NODDAT$),0%)
```

Log User Event: NTEV (DECnet)

The NTEV call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

Note

> This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet/E and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the experienced network user and is not intended for normal network programs.
>
> You need SYSIO and SEND privileges to use NTEV.

BASIC-PLUS-2 Call Format

```
CALL NTEV BY REF(errval%,evmod%,evcls%,evtyp%,entyp%,
                 enval%,buffer$,buflen%,bufoff%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTEV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

evmod

This integer argument specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with a status return of NOROOM (*errval* = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a "missed event" event message, and the call terminates with a successful status (*errval* = 0).

Note that in the second case, the user data is not logged. The system only logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet/E internal events when the event queue is full or there are no more small buffers.

evcls

This integer argument specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480.

evtyp

This integer argument specifies the type of event within the class specified by *evcls*. The event type must be in the range of 0 to 31.

entyp

This integer argument specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

-1	No specific entity
0	Node
1	Line
3	Circuit

enval

This integer argument associates a value to the entity specified by the *entyp* argument. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

buffer

This argument defines the starting address of the buffer containing the optional user data to be processed with the event. If a nonzero offset is specified (see *bufoff*), the offset is added to the starting address of the buffer.

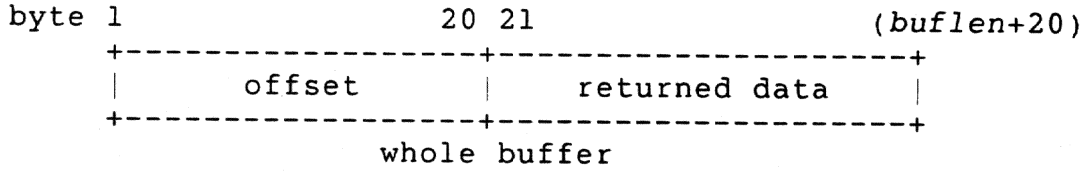
If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. You can use DECnet parameter types if the event concerns a node, a circuit, or a line. In this case, the event logger outputs standard descriptive text for the specific parameter. If you do not use a DECnet parameter type, parameter types in the range of 3900 to 4095 can be specified. (See the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for further information on event definitions and parameters.) Up to 200 bytes of data can be passed to the event logger for processing.

buflen

This integer value defines the number of bytes of data in the buffer specified with *buffer*. Up to 200 bytes of data can be passed. If no data is specified, *buflen* must be 0.

bufoff

This integer value defines an offset, in bytes, from the address specified in *buffer*. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if *bufoff* = 20, the data to be processed begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *buffer*-100.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod*=0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.
10	PROTECTION VIOLATION The calling program did not have SYSIO or SEND privileges at the time the NTEV call was issued.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS() USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet/E was not installed.The network functions cannot be executed.

Example

The following example shows the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of a Local Link Address (*lla*) parameter number (2130) and value. A Missed event message is logged if the event itself cannot be. For example:

```
100 MAP (BUFFER) EVTDAT$ = 200% &
    \ MAP (BUFFER) PARMTY%,DATTYP$=1%,DATVAL$=197%
      .
      .
      .
200 EVMOD% = 1% &
    \ EVCLS% = 480% &
    \ EVTYP% = 3% &
    \ ENTYP% = 0% &
    \ ENVAL% = 135%
      .
      .
      .
300 PARMTY% = 2130% &
    \ DATTYP$ = CHR$(2%) &
    \ PVAL% = 40960% &
    \ DATVAL$ = CHR$(PVAL% AND 255%)+CHR$(SWAP%(PVAL%) AND 255%)
      .
      .
      .
400 ERRVAL% = 0% &
    \ CALL NTEV BY REF(ERRVAL%,EVMOD%,EVCLS%,EVTYP%,ENTYP%, &
      ENVAL%,EVTDAT$,LEN(EVTDAT$),0%)
```

The NTEV call in the previous example generates the following event message:

```
Event type 480.3
Occurred 24-Sep-86 15:31:39.7 on node 3.135 (EDERE)
Node address 3.135 (EDERE)
Parameter #2130 = 40960
```

Send Calls

There are ten separate send calls -- two local and eight network. Each of these calls causes NSP to format a message that is then either queued to a local program or transmitted across the network to a remote program.

Send Local Data Message: MSLD (Local)

The MSLD call sends up to 532 bytes of user data to another program on the local RSTS system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege or the MSLD fails with an error (*errval* = 10). The receiving program is identified in the MSLD call either by its job number or by the logical name with which it declared itself a receiver.

BASIC-PLUS-2 Call Format

```
CALL MSLD BY REF(errval%,job%,name$,buffer$,buflen%,
                 bufoff%,parptr%,parlen%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MSLD call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail.

Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The logical *name* identifies the intended local receiver when the *job* argument = 0. The receiver must have declared its identity with this name in its Declare Receiver call. It is a string containing 1 to 6 characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name must be left-justified and padded with spaces, if necessary. (A store into a 6-byte string defined in COMMON or MAP does this automatically.) Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

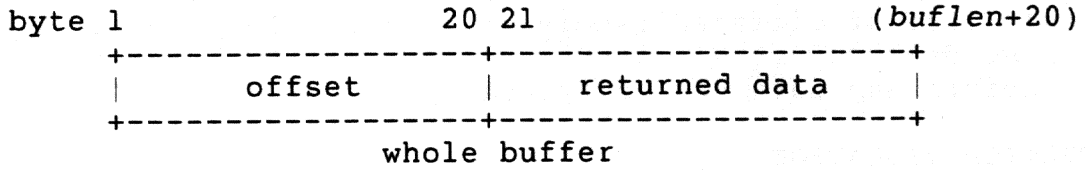
This argument defines the starting address of a buffer from which message data is to be sent. The data is assumed to be in contiguous bytes in the buffer. See the section "Coding" for a discussion of different ways to set up buffers for message services in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*) the offset is added to the starting address specified by *buffer*.

buflen

This integer argument defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the message data to be sent. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSLD call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument defines an area containing up to 20 bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the buffer area. For example, a RSTS BASIC-PLUS-2 program issuing a Receive call that returns a Local Data Message gets the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This integer value defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 20.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS( ) USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512; the *parlen* argument can range from 0 through 20.
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Example

The following example shows a local send to a program that has issued a Declare Receiver call with the logical name LOCAL. A buffer length (*buflen* argument) of 8 and offset of 18 indicate that the characters 'OFFSET' are sent. No parameter message data is sent. For example:

```
1000 COM A$ = 512% &
 \COM LNAME$ =6% &
 \ERRVAL% = 0% &
 \A$ = "TO ILLUSTRATE THE 'OFFSET'" &
 \LNAME$ = "LOCAL" &
 \BUFLEN% = 8% &
 \BUFOFF% = 18% &
 \CALL MSLD BY REF(ERRVAL%,0%,LNAME$,A$,BUFLEN%, &
  BUFOFF%,0%,0%)
```

Send Local Data Message with Privilege Mask: MSPR (Local)

The MSPR call sends up to 524 bytes of user data, plus a privilege mask supplied by the monitor, to another program on the local RSTS system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege or the MSPR fails with an error (*errval* = 10). The receiving program is identified in the MSPR call either by its job number or by the logical name with which it declared itself a receiver.

BASIC-PLUS-2 Call Format

```
CALL MSPR BY REF(errval%,job%,name$,buffer$,buflen%,
                 bufoff%,parptr%,parlen%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MSPR call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver

only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The logical *name* identifies the intended local receiver when the *job* argument = 0. The receiver must have declared its identity with this name in its Declare Receiver call. It is a string containing 1 to 6 characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name must be left-justified and padded with spaces, if necessary. (A store into a 6-byte string defined in COMMON or MAP does this automatically.) Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

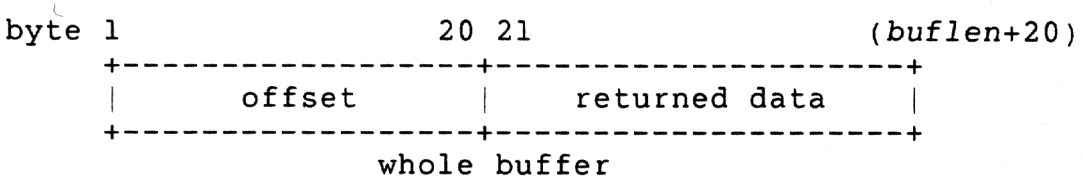
This argument defines the starting address of a buffer from which message data is to be sent. The data is assumed to be in contiguous bytes in the buffer. See the section "Coding" for a discussion of different ways to set up buffers for message services in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*) the offset is added to the starting address specified by *buffer*.

buflen

This integer argument defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the message data to be sent. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSPR call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument defines an area containing up to 12 bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the buffer area. For example, a RSTS BASIC-PLUS-2 program issuing a Receive call that returns a Local Data Message gets the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This integer value defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 12.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	ILLEGAL SYS() USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512; the *parlen* argument can range from 0 through 12.
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Send Connect Initiate Message: NTCI (DECnet)

The NTCI call requests a logical link to another program. The other program is identified with a Connect Data Block and the user link address which the calling program refers to when this link is established. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 bytes of user data can be sent with the NTCI call.

BASIC-PLUS-2 Call Format

```
CALL NTCI BY REF(errval%,ula%,llmod%,rmax%,buffer$,
                 buflen%,bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is an integer value within the range 1 to 255. Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI is accepted by the remote ECL and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

llmod

The integer value *llmod* indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

0 = No flow control
1 = Segment flow control
2 = Message flow control

Chapter 4 describes flow control in detail. Remember that each program in a logical link selects its own flow control. The *llmod* value is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces, can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, NSP supplies the size of its receive buffers as a default value. Note that the calling program is informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive (MRCV)" for a description of how to process large messages in small pieces.

buffer

This argument defines the starting address of a buffer for the Connect Data Block and optional user message data. The 172-byte Connect Data Block and up to 16 bytes of message data are assumed to be in the buffer in contiguous bytes. See the section "Coding" for information on how to set up a buffer in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

The Connect Data Block identifies the target program for the requested logical link. Figure 6-1 shows the format of the Connect Data Block. Table 6-3 describes the fields within the block.

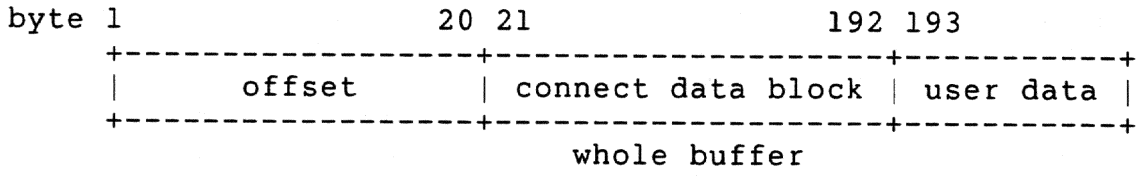
Up to 16 bytes of user data can follow the Connect Data Block.

buflen

This integer value defines the number of bytes to be sent. Since the Connect Data Block is 172 bytes long and the optional user data can be from 0 to 16 bytes long, acceptable values for *buflen* range from 172 through 188.

bufoff

This integer value defines an offset, in bytes, from the start of the buffer from which the data is to be sent. The offset value is added to the starting address of the buffer to define where the Connect Data Block begins. For example, if *bufoff* = 20, the Connect Data Block begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. For example, if a program's buffer is 100 bytes long with a specified offset of 20 and a length of 136, 136 bytes are sent; that is, the last 80 bytes of the buffer and 56 bytes of whatever is beyond the buffer in memory.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

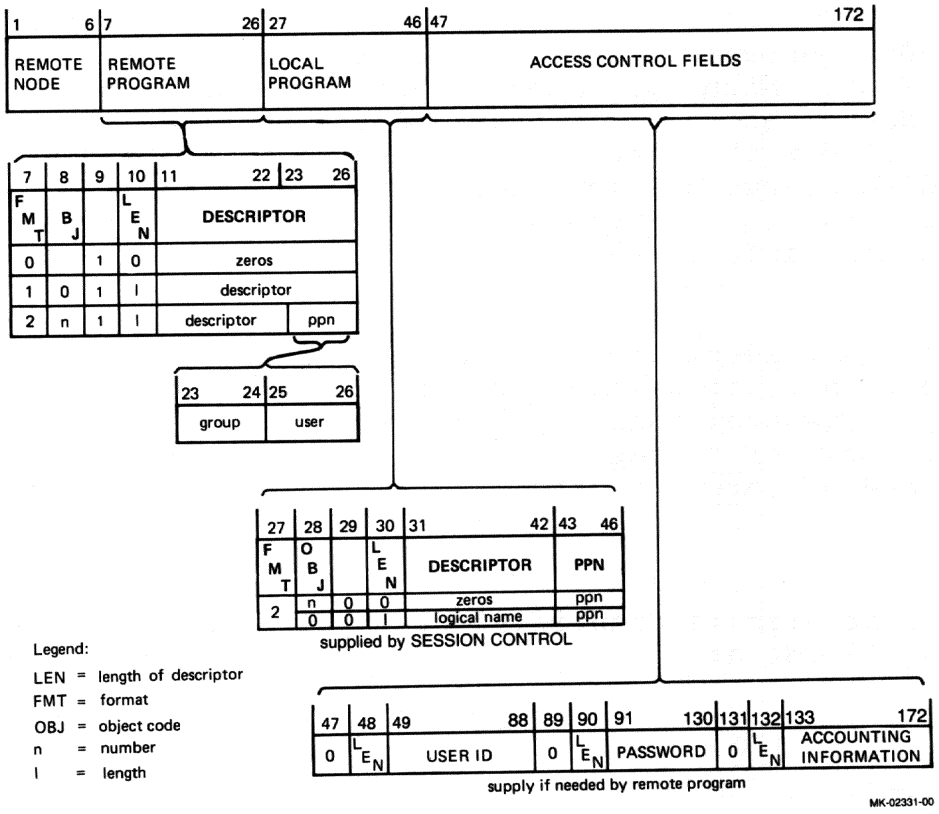


Figure 6-1: Format of Connect Data Block on Send Connect Initiate Message

Table 6-3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node to which the Connect Initiate Message is to be sent. Names shorter than 6 characters must be left-justified with space fill to 6 characters. Leading and embedded spaces are invalid. If no node name is specified (that is, the request is directed to a program at the local node), these 6 bytes contain an ASCII 0 and 5 spaces.
7-26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats: **Format 0** The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9	Must be one.
10-26	Zero (Not used).

Table 6-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	Format 1 The remote program is identified by a descriptor. The fields are: **Bytes** — **Content** 7 — One, to indicate a format 1 name. 8 — Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). 9 — Must be one. 10 — Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field. 11-26 — Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations. **Format 2** This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

Table 6-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The subfields are:
	Bytes: 7 — Two, to indicate a format 2 name.
	Bytes: 8 — Object type code of the remote program, if applicable. If zero, the descriptor must be used.
	Bytes: 9 — Must be one.
	Bytes: 10 — Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
	Bytes: 11-22 — Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
	Bytes: 23-24 — Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E PPN.
	Bytes: 25-26 — Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E PPN.
27-46	**Local Program.** This field is filled in by the local session control using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

Table 6-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with an object code alone, the object code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The RSTS/E PPN under which the program is running is passed in bytes 43-46 as the group and user codes: the project number is in byte 43; the programmer number is in byte 45. Both are in binary format.
47-172	**Access Control Fields.** These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number or system password, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved -- must be zero.
48	Length of user identification information specified in the next field, 0 to 40 bytes.

Table 6-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
49-88	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, set the length (byte 48) to zero.
89	Reserved -- must be zero.
90	Length of password information specified in the next field, 0 to 40 bytes.
91-130	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, set the length (byte 90) to zero.
131	Reserved -- must be zero.
132	Length of the accounting information specified in the following field, 0 to 40 bytes.
133-172	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE The calling program already has a logical link with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NOT A VALID DEVICE The node named in the Connect Data Block is not known to the system.
14	DEVICE HUNG OR WRITE LOCKED The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	TOO MANY OPEN FILES ON UNIT The total number of logical links allowed for the local node has been reached. The system manager can set a limit on the total number of links. See the system manager if this limit is inadequate. A retry may succeed.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS( ) USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Issue a Remove Receiver call and reissue the Declare Receiver call.
- An invalid value was detected in one of the following fields:
  - Remote program field in the Connect Data Block.
  - One of the three access control fields in the Connect Data Block.
  - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.
  - The local link modifier (*llmod*) value must be 0, 1, or 2.
  - The remote descriptor length (byte 10 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).

22 DISK PACK IS LOCKED OUT

Session Control cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. The length of the buffer must be between 172 bytes and 188 bytes for a Connect Initiate Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a Connect Initiate Message directed to a program named NETRCV at node MIAMI. The Connect Data Block uses a format 1 name to address the remote program. Note that the use of the SWAP% function to place a value of 6 in byte 10 of the Connect Data Block. ECL supplies the local program identification and no access control fields are used. Thus, the rest of the Connect Data Block is filled with zeros. Message data is sent following the Connect Data Block. Note that 16 bytes of user message data are actually sent. Storing into a string field defined by MAP causes space fill in the field. Thus, the LEN function returns a value of 188. For example:

```
1000 MAP (SNDCI) CONDAT$ = 188% &
 \MAP (SNDCI) NODE$ = 6%,FMT%,LGT%,NAME$=16%, &
      FILL$(146%)=CHR$(0%),MSG$=16% &
  .
  .
  .
 \ULA% = 27% &
 \LLMOD% = 2% &
 \RMAX% = 512% &
 \NODE$ = "MIAMI" &
 \FMT% = 1% &
 \LGT% = SWAP%(6%) &
 \NAME$ = "NETRCV" &
 \MSG$ = "OPTION2" &
 \RIB% = 17%
  .
  .
  .
 \CALL NTCI BY REF(ERRVAL%,ULA%,LLMOD%,RMAX%, &
      CONDAT$,LEN(CONDAT$),0%,RIB%)
```

Send Connect Confirm Message: NTCC (DECnet)

The NTCC call accepts a remote program's request for a logical link. The call establishes the user link address that the calling program uses to refer to the link. The local link address assigned by the local ECL identifies the particular link being accepted. The flow control option and the receive maximum for the link are specified and up to 16 bytes of message data can be passed to the remote program.

BASIC-PLUS-2 Call Format

```
CALL NTCC BY REF(errval%,ula%,lla%,llmod%,rmax%,
                 buffer$,buflen%,bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCC call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is an integer value within the range 1 to 255. Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The Local Link Address (*lla*) is an integer value that identifies the link being accepted. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in bytes 5-6 of the received Connect Initiate Message.

llmod

The integer value *llmod* indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 = No flow control
1 = Segment flow control
2 = Message flow control

Chapter 4 describes flow control in detail. Remember that both programs in a logical link select their own flow control, independent of each other. The *llmod* value is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, the local ECL supplies the size of its receive buffers as a default value.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument for the Receive call (MRCV) in the section "Receive (MRCV)" for a description of how to process large messages in small pieces.

buffer

This argument defines the starting address of a buffer for the optional user message data. The data is assumed to be in the buffer in contiguous bytes. The section "Coding" for a discussion of how to set up buffers for message operations in BASIC-PLUS-2.) If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

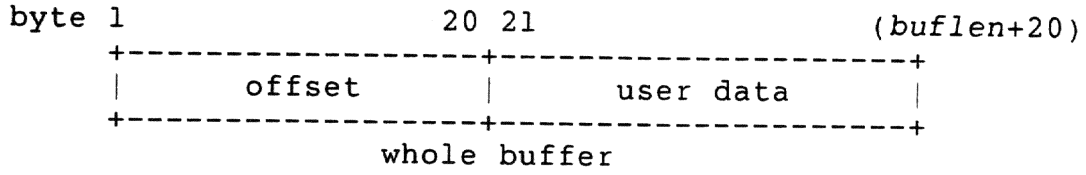
buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The offset value is added to the starting address of the buffer to define where the user data begins.

For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE

A logical link is currently active for the calling program with the same User Link Address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.

5 CAN'T FIND FILE OR ACCOUNT

The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Confirm Message for the same logical link causes this error because the link is already established.

18 ILLEGAL SYS( ) USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- An invalid value was detected in one of the following fields:
  - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.
  - The local link modifier (*llmod*) value must be 0, 1, or 2.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Confirm Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, an NTCC call responds to a received Connect Initiate Message that was stored in the receiving string REC$. The Local Link Address is stored in bytes 5 and 6 of such a string and the second MAP statement has been used so LLA% obtains this integer value. The call also establishes a user link address of 15, a receive maximum of 256 bytes, and indicates that the program requests message flow control. No user message data is sent, so a null dynamic string and zero values are used for the buffer arguments. For example:

```
1000 MAP (RETBUF) REC$ = 40% &
 \MAP (RETBUF) FILL$ = 4%,LLA%,FILL%,REMAINDER%, &
      FILL%,LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%, &
      RMAX%,TMAX% &
   .
   .
   .
 \ERRVAL% = 0% &
 \ULA% = 15% &
 \LLMOD% = 2% &
 \RMAX% = 256% &
 \RIB% = 17%
 \CALL NTCC BY REF(ERRVAL%,ULA%,LLA%,LLMOD%, &
      RMAX%,NOBUF$,0%,0%,RIB%)
```

Send Connect Reject Message: NTCR (DECnet)

The NTCR call rejects a remote program's request for a logical link. The link is identified by the local link address established by the local ECL. You can send up to 16 bytes of user message data to the remote program as part of the Connect Reject Message.

BASIC-PLUS-2 Call Format

```
CALL NTCR BY REF(errval%,lla%,reason%,buffer$,buflen%, bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. If the NTCR call succeeds, this variable is set to zero. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

lla

The Local Link Address (*lla*) is an integer value that identifies the link being rejected. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The *lla* is passed to the local program in bytes 5-6 of a received Connect Initiate Message.

reason

DECnet/E NSP lets a program specify one of the following three reason codes: 0, 34, or 36.

Most user programs use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

buffer

This argument defines the starting address of the buffer containing the optional user message data. The data is assumed to be in the buffer in contiguous bytes. See the section "Coding" for a discussion of how to set up buffers for message operations. If a nonzero offset

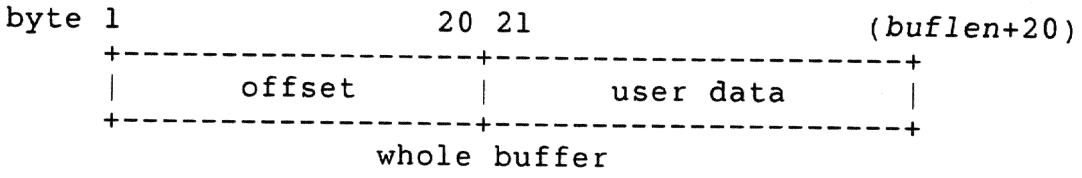
is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT

The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Reject Message for the same logical link returns this error.

18 ILLEGAL SYS( ) USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *reason* code supplied with the Connect Reject Message was not 0, 34, or 36.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Reject Message.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A retry may succeed.

62 NO RUN-TIME SYSTEM

DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

66 MISSING SPECIAL FEATURE

DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, a NTCR call responds to a received Connect Initiate Message stored in the receiving buffer R$. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data. For example:

```
1000 MAP (RETBUF) R$ = 40% &
 \MAP (RETBUF) FILL$ = 4%,LLA%,FILL%,REMAINDER%,FILL%, &
      LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX% &
 \ERRVAL% = 0% &
 \BUFFER$ = "TOO LATE" &
 \BUFLEN% = 8% &
 \BUFOFF% = 0% &
 \REASON% = 0% &
 \RIB% = 17% &
  .
  .
  .
 \CALL NTCR BY REF(ERRVAL%,LLA%,REASON%,BUFFER$, &
      BUFLEN%,BUFOFF%,RIB%)
```

Send Network Data Message: NTDM (DECnet)

The NTDM call transmits user data over an established logical link. The amount of user data you can transmit in one Network Data Message is called a segment. You can also set a flag byte to indicate if the segment is the end of a logical message. If the remote program requested segment or message flow control when the link was established, requests for data must be outstanding or this call results in an error.

BASIC-PLUS-2 Call Format

```
CALL NTDM BY REF(errval%,retbuf$,ula%,dmflgs%,buffer$,
                 buflen%,bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTDM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. You must specify this argument regardless of whether link status information is requested and it should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The *dmflgs* argument is an integer value that can be coded as the sum of three bit values:

```
dmflgs% = rls% + eom% + bom%
```

The low-order two bits of the *dmflgs* argument (*eom%* + *bom%*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, ECL uses these two bits to regulate the transmission of logical messages as the remote program requests them. See Chapter 4

for a full discussion of flow control.

eom	*bom*	Meaning
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E ECL uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is passed to the remote system and program.

ECL also uses the *eom* bit to determine whether it permits a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* bit set, ECL assumes that the last message has not been completely sent and rejects a Disconnect Message. This is always done, without regard to the flow control option selected by the remote program.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, the *eom* flag should always be set. Setting the bit in this case does not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

The *rls* bit is used to request that link status information be returned to the *retbuf* argument when this call is successfully completed:

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message. The MAP statement is particularly useful to allocate the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see the section "Coding"). If no link status information is requested, ECL returns random data.

buffer

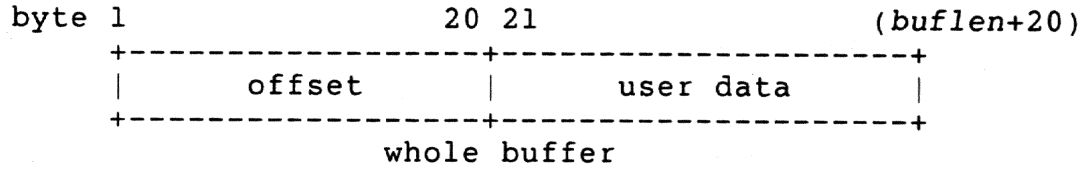
This argument defines the starting address of a buffer containing the user message data. The data is assumed to be in contiguous bytes. See the section "Coding" for a discussion of how to set up buffers in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes to be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or remote ECL. It is passed on to the local program as the transmit maximum -- the *tmax* value in a received Connect Initiate or Connect Confirm Message. Thus, the value of *buflen* can range from 1 through *tmax*.

bufoff

This integer value defines an offset, in bytes, from the starting address of the buffer defined by the buffer argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. One hundred bytes beginning at *buffer*+20 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE

The Data Message transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.

5 CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed causes this error.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be in the range of 1 to 255.

19 DISK BLOCK IS INTERLOCKED

Either the remote system has inhibited transmission on this link because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link).

The calling program is notified with a Link Service Message when the condition clears (see Chapter 4).

Possible Errors (Cont.)

errval	Meaning
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a variable-length send from a buffer area defined by a dynamic string. The LEN function returns the correct number of bytes for the length argument. The ULA is 20. The value for DMFLGS% identifies the message data as the sole segment of a logical message and requests the return of link status information. Note the use of MAP statements to allocate space for the buffer for the returned data. First, the buffer for returned data is set up as 40 bytes long. Next, a MAP is made with a useful format for returned link status information. (Other MAPs of the same area would presumably be made for other message types. They are not shown here since the concern is only with returned link status.) After the call, the program checks for an error. If none occurred, the program checks to see if there are any pending messages. If so, the program branches to receive and process messages. If not, the program continues.

continues. Note the use of the ASCII function to get integer values for the returned data. For example:

```
1000        MAP (RETBUF) A$ = 40% &
     \MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=16%, &
          LSTAT$=1%,RSTAT$=1%,LDR$=1%,&
          LIR$=1%,RDR$=1%,RIR$=1%,DTM$=1%,DTC$=1%,LTM$=1%, &
          LTC$=1%,MMAX$=1%,MCNT$=1%,MULA$=1% &
     \EOM% = 2% &
     \BOM% = 1% &
     \RLS% = 128% &
     \RIB% = 17% &
           .
           .
           .
     \B$ = "DYNAMIC STRING SEND -- LEN WILL RETURN LENGTH" &
     \ERRVAL% = 0% &
     \ULA% = 20% &
     \DMFLGS% = EOM%+BOM%+RLS% &
     \CALL NTDM BY REF(ERRVAL%,A$,ULA%,DMFLGS%,B$, &
                       LEN(B$),0%,RIB%) &
     \IF ERRVAL% <> 0% THEN GOTO 1500 &
     \IF ASCII(MCNT$) > 0 THEN GOTO 4000 &
           .
           .
           .
```

Send Interrupt Message: NTIN (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message is placed at the head of the pending message queue, behind the first message (since it can already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt Messages are subject to flow control (see Chapter 4).

BASIC-PLUS-2 Call Format

```
CALL NTIN BY REF(errval%,retbuf$,ula%,inflgs%,buffer$,
                 buflen%,bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTIN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. You must specify this argument regardless of whether link status information is requested and it should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

The *inflgs* argument indicates whether link status information is to be returned to *retbuf* when the call has completed successfully. This field should be coded as follows:

```
inflgs% = rls%
```

where:

```
rls = 128 Link status information is returned.
rls =   0 No link status information is returned.
```

If link status information is requested, it is returned in the same format as a received Link Service Message. The MAP statement is particularly useful to allocate the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see the section "Coding").

If no link status information is requested, ECL returns random data.

buffer

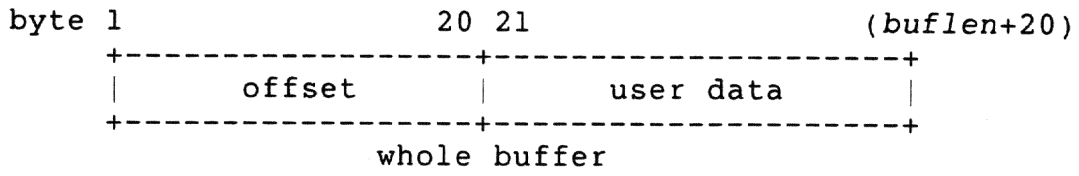
This argument defines the starting address of a buffer containing optional user message data. See the section "Coding" for information on how to set up a buffer in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer to determine where the data begins.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a bufoff of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255.

RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed causes this error.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS( ) USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.
19	DISK BLOCK IS INTERLOCKED The remote ECL has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message is delivered to the calling program when the condition clears. See Chapter 4 for a full discussion of flow control.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for an Interrupt Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTIN call for logical link 15 on RIB #17. The call sends the message data TIMEOUT and requests no link status information to be returned. A buffer must be defined and referenced, however, so a MAP is set up with the string name DUMMY$ for reference in this case. (The same buffer area can be used with different mappings for other calls. The name DUMMY$ documents that the data returned is random and not used by the program.) For example:

```
1000     MAP (RETBUF) DUMMY$ = 40% &
                .
                .
                .
        \ERRVAL% = 0% &
        \A$ = "TIMEOUT" &
        \BUFLEN% = LEN(A$) &
        \OFFSET% = 0% &
        \INFLGS = 0% &
        \ULA% = 15% &
        \RIB% = 17% &
        \CALL NTIN BY REF(ERRVAL%,DUMMY$,ULA%,INFLGS%, &
                          A$,BUFLEN%,BUFOFF%,RIB%)
```

Send Link Service Message: NTLS (DECnet)

The NTLS call:

- Requests data from a remote program over a flow-controlled logical link (see Chapter 4)
- Reenables incoming interrupts over a logical link (see Chapter 4)
- Obtains status information for the link

No user data is allowed with a Link Service Message.

BASIC-PLUS-2 Call Format

```
CALL NTLS BY REF(errval%,retbuf$,ula%,lsflgs%,drcnt%, ircnt%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLS call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to receive link status information returned by the call. You must specify the argument regardless of whether link status information is requested and it should refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The link service flags argument is coded as the sum of two values: *lsflgs%* = *rls%* + *opr%*. The low-order two bits (*opr*) indicate the purpose of the call:

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. You can only use this form of the call when the calling program requested message or segment flow control in its Connect Initiate or Connect Confirm Message for the logical link. If no flow control was requested, an NTLS call with *opr* = 0 fails, returning an error (*errval* = 18).

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* argument is interpreted as the number of Interrupt Messages requested (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

You can use the *rls* bit when *opr* = 0 or 1 to request that link status information be returned to the return buffer (defined by *retbuf*) when the call completes successfully:

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

Link status information is always returned when *opr* = 2. If requested, the information returned is in the same format as a received Link Service Message. The MAP statement is particularly useful to define the buffer, as it both allocates the space for the information and allows the definition of named fields with different data types within the buffer (see the section "Coding").

If no link status information is requested, random data is returned.

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpreted as either a segment counter (segment flow control) or a logical message counter (message flow control). In either case, it is an incremental count, added to the counter of outstanding requests kept by the remote and local ECL (see Chapter 4). The program returns an error if the *drcnt* given increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is meaningful only when *opr* = 1 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Thus, *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling interrupts on the logical link. Note that you can only reenable interrupts when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call fails with an error (*errval* = 19).

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. ECL is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.

Possible Errors (Cont.)

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

10	PROTECTION VIOLATION

Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed causes this error.

18	ILLEGAL SYS() USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *opr* value is not 0, 1, or 2.
- The *opr* value is 0, but the program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message.
- The data request count (*drcnt*) increased the count of outstanding segments or logical messages requested to more than 127.
- The *opr* value is 1, indicating a request for an Interrupt Message, but the *ircnt* argument does not equal 1.
- The *opr* value is 1, indicating a request for an Interrupt Message, but the outstanding request counter for Interrupt Messages already equals 1.
- The *ula* specified is zero. It must be within the range 1 to 255.

Possible Errors (Cont.)

errval	Meaning
19	DISK BLOCK IS INTERLOCKED The *opr* value is 1, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

This example of the NTLS call requests five segments from the remote program over logical link 23 on RIB #17. Assume that the local program requested segment flow control in its NTCI or NTCC call for this link. For example:

```
1000    MAP (RETBUF) A$ = 40% &
        \OPR% = 0% &
        \RLS% = 128% &
         .
         .
         .
        \ERRVAL% = 0% &
        \ULA% = 23% &
        \LSFLGS% = RLS%+OPR% &
        \DRCNT% = 5% &
        \RIB% = 17% &
        \CALL NTLS BY REF(ERRVAL%,A$,ULA%,LSFLGS%,DRCNT%,0%,RIB%)
```

Send Disconnect Message: NTDI (DECnet)

The NTDI call terminates an established logical link. Messages in the pending message queue remain but no more messages can be sent over the link. The user link address is freed and can be used for another logical link. If the last Network Data Message sent did not have the end-of-message (EOM) flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call fails with an error.

Successful completion of an NTDI assures the sender that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not guarantee that the receiving program has processed the messages, however. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see the section "Send/Receive Link Service Message" in Chapter 3). Otherwise, the NTDI provides no particular advantage over an NTLA in terminating a logical link (see the section "Send Link Abort Message").

Up to 16 bytes of user data can be passed with the Disconnect Message.

BASIC-PLUS-2 Call Format

```
CALL NTDI BY REF(errval%,ula%,buffer$,buflen%,bufoff%, rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTDI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

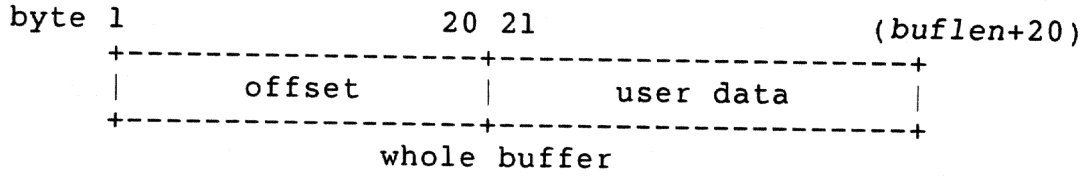
This argument defines the starting address of a buffer containing the optional user message data. See the section "Coding" for information on how to set up buffers in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer argument defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. If, for example, a program's buffer is 20 bytes long, with an offset of 10 and a *buflen* of 16, 16 bytes are sent: the last 10 bytes of the buffer, and 6 bytes of whatever is beyond the buffer in memory.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE

There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)

5 CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed causes this error.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be within the range 1 to 255.

19 DISK BLOCK IS INTERLOCKED

The last segment sent by the local program over this logical link did not have the *eom* flag set. ECL does not permit a logical link to be disconnected unless all messages have been completely sent. Note that this test of the *eom* flag is done irregardless of the flow control option chosen by the local program.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example disconnects logical link 255 on RIB #17 and includes user data explaining the reason for the disconnect. Note that the remote program has to be coded to recognize this message and take appropriate action. For example:

```
\ULA% = 255% &
\RIB% = 17% &
\ERRVAL% = 0% &
\CALL NTDI BY REF(ERRVAL%,ULA%,"NOMORE",6%,0%,RIB%)
```

Send Link Abort Message: NTLA (DECnet)

The NTLA call terminates a logical link immediately. If there are messages in the transmit queues waiting for acknowledgment from the remote ECL, the system discards them. These messages may or may not have been received by the remote system (see the section "Send/Receive Disconnect Message" in Chapter 3).

You can send up to 16 bytes of user data with the message. If you issue the NTLA for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If you issue the NTLA for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program.

BASIC-PLUS-2 Call Format

```
CALL NTLA BY REF(errval%,ula%,buffer$,buflen%,bufoff%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLA call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

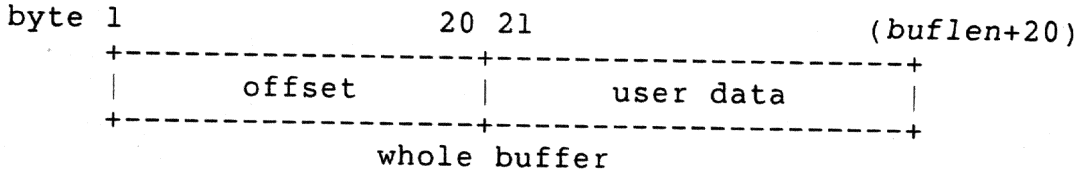
This argument defines the starting address of a buffer where the optional user message data begins. See the section "Coding" for information on how to set up buffers in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the buffer argument.

buflen

This integer value defines the number of bytes of user data to be sent to the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to find the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be in the range of 1 to 255.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example aborts logical link 17 on RIB #17; no user data is sent:

```
1000    MAP (BUFFER) A$ = 256% &
          .
          .
          .
        RIB% = 17% &
        \CALL NTLA BY REF(ERRVAL%,17%,A$,0%,0%,RIB%)
```

Receive: MRCV (Local and DECnet)

The MRCV call retrieves a message from the calling program's pending message queue. It returns control parameters and user data, if any, to buffers defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer that receives the user data. The message data can be retrieved all at once or in portions, and portions can be discarded. The formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message" show what control information is returned for the various types of messages. A Receive call can be selected to get:

- The first message in the queue
- The first message in the queue from either a network or a local sender
- The first message in the queue for a particular logical link

You can set a sleep flag and an associated timer to indicate that the calling program is to wait if there are no appropriate messages pending in the queue. Otherwise, if there are no appropriate messages pending, the Receive call terminates with an error (*errval* = 5).

BASIC-PLUS-2 Call Format

```
CALL MRCV BY REF(errval%,retbuf$,rmod%,sndr%,qual%,buffer$,
                 buflen%,bufoff%,slptime%,rib%)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MRCV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the buffer to contain the control information identifying the type and characteristics of the message received. This argument must refer to a predefined buffer at least 40 bytes long. The MAP statement is particularly useful to define the buffer, as it both allocates the space for the returned information and allows the definition of named fields with different data types within the buffer (see the section "Coding"). The format of the data returned depends on the message type, as described in the following sections, "Format of Received Local Data Message" through "Format of Received Link Abort Message."

rmod

The receive modifier (*rmod*) argument is an integer value consisting of four flags: *s*% + *t*% + *l*% + *n*%, as follows:

s The sleep flag (*s*) defines what is to be done if there is no appropriate message in the queue for the Receive call to deliver -- that is, a message of the type requested with the *l*, *n*, *sndr*, and *qual* arguments. If *s* = 1, the program sleeps, suspending execution until something happens. The *slptime* argument regulates the length of time that the program sleeps.

If the *s* flag is set to 0, the program does not sleep. If there are no appropriate messages in the queue the call terminates immediately with *errval* = 5.

t The truncate flag (*t*) is set to 2 to discard any excess information that does not fit in the buffer specified in this call. Information could be left over if the value of *buflen* is smaller than the amount of user data in the message.

A 0 value for *t* indicates that such excess information is to be kept for retrieval on later Receive calls.

l The local sender flag (*l*) is set to 4 to select a message from a local sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *l* = 0, local selection is not requested. If both *l* and *n* are zero, the first message in the queue is returned.

n The network sender flag (*n*) is set to 8 to select a message from a network sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *n* = 0, network selection is not requested. If both *l* and *n* = 0, the first message pending in the queue is returned. If *l* = 4 and *n* = 8, local selection prevails.

sndr

The sender selection argument (*sndr*) selects a particular local sender or a particular logical link for this Receive call.

If *l* = 4 (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If *l* = 0 and *n* = 8 (network selection), a nonzero value for *sndr* is interpreted as a ULA. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either *l* = 4 or *n* = 8, the *qual* parameter has special meaning. The *sndr* and *qual* arguments are ignored if both *l* and *n* are zero.

qual

The sender selection qualifier (*qual*) argument is normally zero for user applications.

If *l* = 4 (local selection) and *sndr* = 0, then any nonzero value for *qual* is a special Receive call requesting a message from the system (job number 0). The RSTS/E utility ERRCPY uses this special case to process messages from the monitor error logging routines.

The *qual* argument is ignored if both *l* and *n* = 0, or if *sndr* ≠ 0.

Table 6-4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 6-4: Sender Selection Summary

rmod				
n	*l*	*sndr*	*qual*	Result
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	4	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times two. Only messages from that job are delivered on this MRCV call.

Table 6-4: Sender Selection Summary (Cont.)

rmod				
n	*l*	*sndr*	*qual*	Result
8	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a ULA. Only network messages from the designated logical link are delivered on this MRCV call.

buffer

This argument defines the starting address of the buffer to which user message data is to be delivered. See the section "Coding" for a discussion of how to set up buffers in BASIC-PLUS-2. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the buffer argument.

buflen

This integer argument defines the maximum number of bytes to be delivered to the buffer on this Receive call.

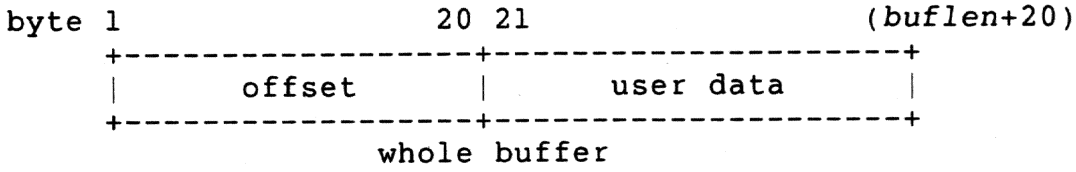
If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (*t* flag of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, then only the amount indicated by *buflen* is returned and the rest is saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining bytes 9-10 in the data returned to the *retbuf* buffer by the Receive call.

The number of bytes actually returned to the buffer is returned in bytes 13-14 in the *retbuf* buffer. See the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the buffer argument. The value of *bufoff* is added to the starting buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user message data is delivered beginning in byte 21 of the buffer.



Note

The BASIC-PLUS-2 interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. Up to 100 bytes beginning at *buffer*+20 could be delivered on the Receive call.

slptime

The *slptime* argument is significant only when the *s* flag of *rmod* = 1. In that case, a positive value for *slptime* defines the length of time (in seconds), that a program is to sleep when no appropriate message is in the queue. The program sleeps until:

- The sleep timer expires.
- Any new message is placed in the queue (not just an appropriate one).
- A delimiter is typed on a terminal opened by, or assigned to, the job.
- Log-ins are disabled. (This could occur if the system is being shut down.)
- A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
- The job itself has opened the XM: (DMC11 driver) and a message is received for the job.
- The system time is changed.

In all cases, the program is awakened with an error (*errval* = 5) but is not passed a message. To obtain a pending message the program must execute another Receive call. Since the program can be awakened by any of the conditions listed previously, you can check for pending messages by executing a MRCV without a sleep.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT For a Receive call without sleep (*s* flag in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (*s* flag in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV call to retrieve any pending messages.
18	ILLEGAL SYS() USAGE No RIB exists for the calling program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example

In the following example, the first Receive call is issued for network messages, with the sleep bit set and the timer set to five seconds. The truncate bit is 0 and the length argument is 0, so all the information that is available is saved for the second Receive call. Note that the test for ERRVAL% = 5% after the first Receive call, with sleep, ensures that a message is waiting when the second Receive call is issued. The Receive is done on RIB #17. For example:

```
1000 MAP (RET) RETBUF$ = 40% &
  \MAP (BUF) BUFFER$ = 256% &
  \S% = 1% &
  \N% = 8% &
    .
    .
    .
  \ERRVAL% = 0% &
  \RMOD% = S%+N% &
  \SLPTIM% = 5% &
  \RIB% = 17%
2000 \CALL MRCV BY REF(ERRVAL%,RETBUF$,RMOD%,0%,0%, &
                       BUFFER$,0%,0%,SLPTIM%,RIB%) &
  \IF ERRVAL% = 5% THEN GOTO 2000 &
  \RMOD% = N% &
  \CALL MRCV BY REF(ERRVAL%,RETBUF$,RMOD%,0%,0%, &
                    BUFFER$,256%,0%,0%,RIB%) &
```

Format of Received Local Data Message

The following format describes the information returned to the *retbuf* buffer for a Local Data Message. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive (MRCV) call.

Bytes	Value	Description
1-2	--	Reserved.
3	-1%	The function code for a Local Data Message.
4	*jobx2*%	The job number (of the local sender) times two.
5-6	*ppn*%	PPN of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	*remainder*%	Number of bytes of user data not delivered to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-40	*param*$	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Sample Map

```
MAP (RETBUF)  A$ = 40%
MAP (RETBUF)
FILL%,CODE$=1%,JOBX2$=1%,PROJ$=1%,PROG$=1%,KB$=1%,FILL$=1%, &
              REMAINDER%,FILL%,LGT%,FILL$=6%,PARAM$=20%
```

Format of Received Local Data Message with Privilege Mask

The following format describes the information returned to the *retbuf* buffer for a Local Data Message with Privilege Mask. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive (MRCV) call.

Bytes	Value	Description
1-2	--	Reserved.
3	-11%	The function code for a Local Data Message.
4	*jobx2%*	The job number (of the local sender) times two.
5-6	*ppn%*	PPN of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	*remainder%*	Number of bytes of user data not delivered to I/O buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* in the MRCV call.
11-12	--	Reserved.
13-14	*length%*	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-28	*priv$*	The sender's privilege mask.
29-40	*param$*	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 12 bytes.

Sample Map

```
MAP (RETBUF)  A$ = 40%
MAP (RETBUF)
FILL%,CODE$=1%,JOBX2$=1%,PROJ$=1%,PROG$=1%,KB$=1%,FILL$=1%, &
                REMAINDER%,FILL%,LGT%,FILL$=6%,PRIV$=8%,PARAM$=12%
```

Format of Received Connect Initiate Message

The following format describes the information returned to the *retbuf* buffer for a received Connect Initiate Message. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the data buffer specified in the Receive call. The format of a received Connect Data Block is shown in Figure 6-2 and described in detail in Table 6-5.

Bytes	Value	Description
1-2	--	Reserved.
3	-2%	The function code for a Connect Initiate Message.
4	--	Reserved.
5-6	*lla*%	Local Link Address. Identifies the link for the local program's subsequent Connect Confirm or Connect Reject Message.
7-8	--	Reserved.
9-10	*remainder*%	The number of bytes of data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23	CHR$(*rlmod*%)	Remote link modifier. Indicates the flow control in effect for the remote program. The low-order two bits should be masked and interpreted as follows: *flow*% = *rlmod*% AND 3% If *flow* = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.

Format of Received Connect Initiate Message (Cont.)

Bytes	Value	Description
25-26	*rmax*%	Receive maximum, in bytes. The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This value is supplied by the local ECL and is determined by the size of the receive buffers it allocates. The size of the receive buffers is set for ECL by Network Management. The local program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a logical link.
27-28	*tmax*%	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40
MAP (RETBUF) FILL%,CODE$=1%,FILL$=1%,LLA%,FILL%,REMAINDER%, &
             FILL%,LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX%
```

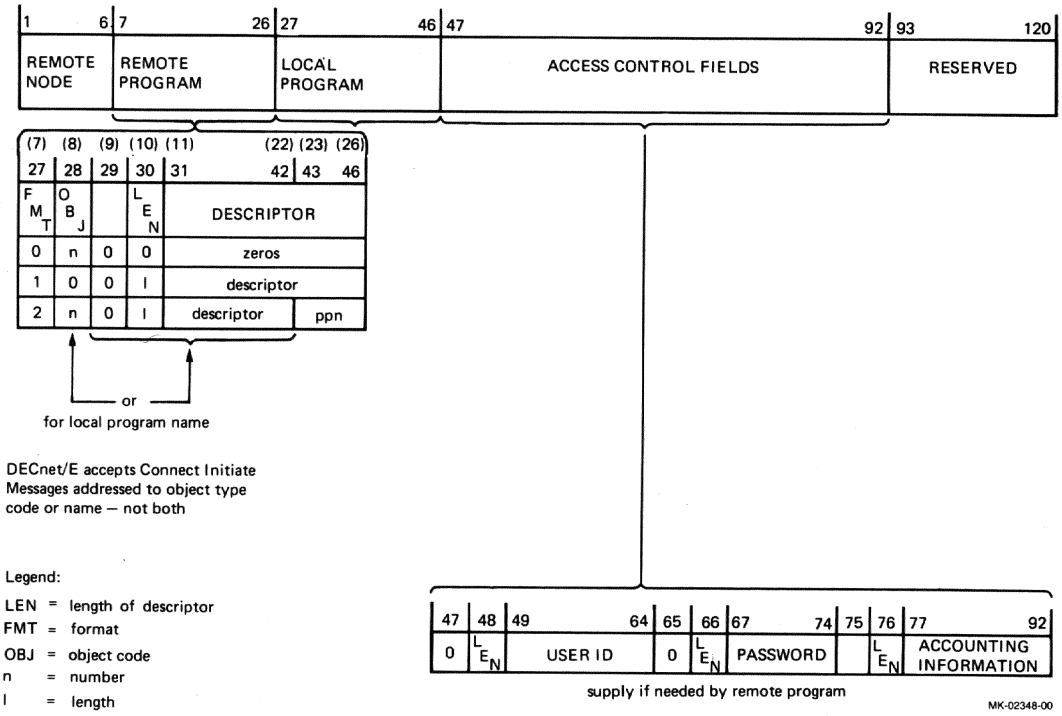


Figure 6-2: Format of Received Connect Data Block

Table 6-5: Format of Received Connect Data Block

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node from which the Connect Initiate Message was received. Names shorter than 6 characters are left-justified with space fill to 6 characters.
7-26	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of the following three formats.

Table 6-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 0 The remote program is identified by object type code alone. The fields are: **Bytes** — **Content** 7 — Zero, to indicate a format 0 name. 8 — Object type code of the remote program. 9-26 — Zero (not used). **Format 1** The remote program is identified by a descriptor. The subfields are: **Bytes** — **Content** 7 — One, to indicate a format 1. 8 — Normally zero. This byte is usually zero unless the target DECnet system identifies the remote program with both name and object type code on outgoing connect requests. 9 — Reserved. 10 — Length (in bytes) of the descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26). 11-26 — Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Table 6-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 2 This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC. DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved.
10	Length (in bytes) of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor is left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E PPN.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E PPN.

Table 6-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
27-46	**Local Program Name.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Equal to 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.

Table 6-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
47-92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most precise checking done in a normal login on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system passes this information on to the receiving program. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49-64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN at login.
65	Reserved.
66	Length of password information specified in the next field, 0 to 8 bytes.
67-74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login.

Table 6-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	75 Reserved.
	76 Length of the accounting information specified in the following field, 0 to 16 bytes.
	77-92 Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.
93-120	Reserved.

Sample MAP of Connect Data Block

```
MAP (CDB) A$ = 120%
MAP (CDB) REMNODE$= 6%,REMFMT$=1%,REMOBJ$=1%, &
    FILL$=1%,REMLEN$=1%,REMNAME$=12%, &
    REMGRP$=1%,FILL$=1%,REMUSR$=1%,FILL$=1%, &
    LOCFMT$=1%,LOCOBJ$=1%,FILL$=1%, &
    LOCLEN$=1%,LOCNAME$=12%, &
    LOCGRP$=1%,FILL$=1%,LOCUSR$=1%,FILL$=1%, &
    FILL$=1%,IDLGT$=1%,USRID$=16%, &
    FILL$=1%,PWDLGT$=1%,PASSWRD$=8%, &
    FILL$=1%,ACCTLGT$=1%,ACCNT$=16%,FILL$=28%
```

Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the local program. The link is identified by the User Link Address specified by the local program in the Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. Up to 16 bytes of user data can also be returned to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-3%	The function code for a Connect Confirm Message.
4	CHR$(*ula*%)	User Link Address, previously defined by the local program in its Connect Initiate Message.
5-8	--	Reserved.
9-10	*remainder*%	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23	CHR$(*rlmod*%)	Remote link modifier. Indicates the flow control in effect for the remote program. The two low-order bits should be masked and interpreted as follows: *flow*% = *rlmod*% AND 3% If *flow* = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.

Format of Received Connect Confirm Message (Cont.)

Bytes	Value	Description
25-26	*rmax*%	Receive maximum, in bytes. The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This is: o The *rmax* value specified in the Connect Initiate Message for this link o A lesser value supplied by the local ECL, calculated from the size of the buffers that it allocates for receives.
27-28	*tmax*%	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=8%,RLMOD$=1%,FILL$=1%,RMAX%,TMAX%
```

Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the local program's request for a logical link. The link is identified by the user link address previously established in the local program's Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. Rejections from ECL contains a reason code explaining the reason for the rejection and does not contain user data. Rejections from the remote program can contain up to 16 bytes of user data, delivered to the data buffer defined in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-4%	The function code for a Connect Reject Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate Message.
5-8	--	Reserved.
9-10	*remainder*%	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23-24	*reason*%	Identifies the reason for the rejection (see Appendix B).
25-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=8%,REASON%
```

Format of Received Network Data Message

A Received Network Data Message contains user data from a remote program. The following format describes the information returned to the *retbuf* buffer. The user data accompanying the Network Data Message is delivered to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-5%	The function code for a Network Data Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder*%	The number of bytes of data not delivered to the data buffer. This data has been either discarded or saved for a later Receive, depending on how the *t* flag was set in the *rmod* in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the data buffer on this Receive call.
15-20	--	Reserved.

Format of Received Network Data Message (Cont.)

Bytes	Value	Description
21	*dmflgs%*	Data message flags. Indicate whether this is the beginning, middle, end, or sole segment of a logical message. The low-order two bits should be masked and interpreted as follows: *bom%* = *dmflgs%* AND 1% *eom%* = *dmflgs%* AND 2% Where: **eom bom** **Meaning** 0 0 Middle segment of message 0 1 First segment of message 2 0 Last segment of message 2 1 Sole segment of message DECnet/E uses the *eom* bit to determine the end of a logical message if the local program requested message flow control when the logical link was established. It does not use the *bom* bit but passes it on to the receiving program.
22-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%,REMAINDER%,FILL%, &
             LGT%,FILL$=6%,DMFLGS$=1%
```

Format of Received Interrupt Message

The local ECL places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The following format describes the data returned to the *retbuf* buffer. Up to 16 bytes of user data can be delivered to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-6%	The function code for an Interrupt Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder*%	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the buffer on this Receive call.
15-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%, &
             REMAINDER%,FILL%,LGT%
```

Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call whenever the pending message queue has emptied after a condition has cleared that previously inhibited the local program from sending (see the section "Flow Control" in Chapter 4). Status information with the same format as a Link Service Message can also be returned to the *retbuf* buffer on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The following format describes the information returned. No data is delivered to the data buffer with a Link Service Message.

Bytes	Value	Description
1-2	--	Reserved.
3	-7%	The function code for a Link Service Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	0%	Always zero for Link Service Messages. (These bytes are the *remainder* field in other returned messages.)
11-12	--	Reserved.
13-14	0%	Always zero for Link Service Messages. (These bytes are the *length* field in other returned messages.)
15-20	--	Reserved.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
21	*lstat%*	Local status flags. These flags should be masked and interpreted as follows: *data%* = *lstat%* AND 1% *intls%* = *lstat%* AND 2% *lclbp%* = *lstat%* AND 192% (2-bit field) If *data* = 1, outgoing Network Data Messages are inhibited. (A Link Service Message is delivered when the condition clears.) If *intls* = 2, outgoing Interrupt and Link Service Messages are inhibited. (A Link Service Message will be delivered when the condition clears.) The *lclbp* value indicates the local ECL's backpressure status: 0 Incoming Data Message flow is on. 64 Incoming Data Message flow is on, but the local ECL turns this link off if another Data Message is received before the local program's receive queue is emptied. 128 Incoming Data Message flow has been turned off by the local ECL. The local program's receive queue must be emptied before the link is turned on. 192 Incoming Data Message flow has been turned off by the local ECL but flow is scheduled to turn on as soon as system buffers become available.
22	*rstat%*	Remote status flags. Only one bit is currently defined for this field. It should be masked and interpreted as follows: *rembp%* = *rstat%* AND 128% If *rembp* = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
23	*ldr%*	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	*lir%*	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of interrupts requested but not yet received. This count will never exceed one.
25	*rdr%*	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	*rir%*	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote program, reflecting the actual count of Interrupt Messages requested but not yet sent.
27	*dtm%*	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.
28	*dtc%*	Data transmit queue count. The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote ECL.
29	*ltm%*	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.

Format of Received Link Service Message (Cont.)

Bytes	Value	Description
30	*ltc%*	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote ECL.
31	*mmax%*	Message maximum. The maximum number of incoming messages that are queued for the local program. This maximum is set by the program in the Declare Receiver call.
32	*mcnt%*	Message count. The total number of messages currently in the pending message queue for the local program.
33	*mula%*	Message count for this logical link. The number of messages currently in the pending message queue for this logical link, as identified by the *ula* (byte 4).
34-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=16%, &
       LSTAT$=1%,RSTAT$=1%,LDR%=1%,LIR$=1%, &
       RDR$=1%,RIR$=1%,DTM$=1%,DTC$=1%, &
       LTM$=1%,LTC$=1%,MMAX$=1%, &
       MCNT$=1%,MULA$=1%
```

Format of Received Disconnect Message

A Received Disconnect Message indicates that an established logical link with a remote program has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The following format describes the data returned to the *retbuf* buffer. Up to 16 bytes of user message data can be returned to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-8%	The function code for a Disconnect Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	*remainder*%	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the data buffer on this Receive call.
15-40	--	Reserved.

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,FILL$=4%, &
      REMAINDER%,FILL%,LGT%
```

Format of Received Link Abort Message

A Received Link Abort Message indicates that a logical link has been terminated. The following format describes the data returned to the *retbuf* buffer. Up to 16 bytes of user message data can be returned to the data buffer specified in the Receive call.

Bytes	Value	Description
1-2	--	Reserved.
3	-9%	The function code for a Link Abort Message.
4	*ula*%	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-6	*lla*%	Local Link Address, established by the local ECL for this link. It identifies the link if the remote program has sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists under these conditions to identify the link being aborted.
7-8	--	Reserved.
9-10	*remainder*%	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	*length*%	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23-24	*reason*%	Abort Reason. If nonzero, the link was aborted by either the local or remote ECL and the reason is specified in these two bytes. See Appendix B for a list of reasons that apply for a link abort. If *reason* is zero, the link was aborted at the request of the remote program.
25-40	--	Reserved.

Format of Received Link Abort Message
BASIC-PLUS-2

Sample Map

```
MAP (RETBUF) A$ = 40%
MAP (RETBUF) FILL%,CODE$=1%,ULA$=1%,LLA%,FILL%, &
      REMAINDER%,FILL%,LGT%,FILL$=8%,REASON%
```

Chapter 7

Overview of the Concise and Extended Interfaces for COBOL

This chapter briefly describes the concise and extended interfaces. The COBOL programmer can access the DECnet/E message services in one of two forms: the concise COBOL interface or the extended COBOL interface. To help you decide which interface to use, the last section summarizes the advantages and disadvantages of each interface.

The Concise COBOL Interface

The design of the concise COBOL interface lets the COBOL programmer regard the network as a file which allows you to:

- Open the file (establish a logical link)
- Send/Receive records (up to 32767 bytes of data in any COBOL-definable category)
- Close the file (disconnect the logical link)

Only one network file can be open at a time; that is, you can establish only one logical link with another program in the network. When established, both programs can send and receive data over the logical link and either program can disconnect the logical link.

The following five calls are available with the concise COBOL interface:

- **Connect Request Call: CNTCON** -- The CNTCON call requests a logical link connection with another program in the network. The call identifies the node and the remote program's name or object type code. ECL forwards the request to the specified node. If a program is running there that has already issued a Connect Accept call, identifying itself with the proper object type code or name, or if a program with the proper identity has been defined for automatic startup, the request for a connection is delivered. You can include information

to permit the remote system or program to verify the calling program's right to use the remote system or program's services. If the remote program does not accept the connection within five minutes, the CNTCON call terminates with an error value returned to an argument specified in the call.

- **Connect Accept Call: CNTACP** -- The CNTACP call accepts a logical link connection request from another program in the network. In the call, the program identifies itself with a network name or object type code. The calling program can also supply information to be used by ECL to check the requestor's rights to access the program's services. If this information does not match that specified by the remote program in its connect request, ECL rejects the request for the logical link.

- **Send Record Call: CNTSND** -- The CNTSND call lets a program send up to 32767 bytes of data to a remote program over an established logical link. As with all RSTS/E input and output, the CNTSND call is synchronous. That is, control returns to the calling program when all the data is sent and has been received by the remote system.

- **Receive Record Call: CNTRCV** -- The CNTRCV call delivers up to 32767 bytes of data to a data area specified by the calling program. Like CNTSND, the CNTRCV call is synchronous.

- **Disconnect Call: CNTDIS** -- The CNTDIS call disconnects the established logical link. Either program can disconnect a logical link. A disconnection causes a CNTRCV issued by the other end of the logical link to terminate with an end-of-file indication.

With the concise COBOL interface, you must design your programs so two possible deadlock situations cannot occur; that is, simultaneous send calls or simultaneous receive calls at both ends of a logical link.

If both programs using a logical link try to send large amounts of data at the same time, and neither program's allotted portion of system buffer space is large enough to hold its incoming record, both programs suspend execution indefinitely, waiting for the I/O operation to complete. Thus, neither program can receive and thereby alleviate the congestion, because neither send can complete.

Similarly, if both programs using a logical link execute a CNTRCV at the same time, without sending data first, both wait indefinitely for data to be received.

In the case where one program only sends data and the other program only receives data, neither of the deadlock situations can ever occur. In the more general case, where both programs can send and receive, a message dialogue (using CNTCON and CNTACP) must be established between the programs so each program recognizes its turn to send or receive.

The Extended COBOL Interface

The extended COBOL interface allows more finely tuned network operations than the concise COBOL interface. A program can establish more than one logical link with the same or separate remote programs. Also, a program can select flow control options to regulate the flow of incoming data. Such a program can be designed so it never waits indefinitely after trying to send or receive.

The extended COBOL interface calls reflect the way in which the network message services are implemented in the RSTS/E monitor. The same functions are available to MACRO-11, FORTRAN, BASIC-PLUS, and BASIC-PLUS-2 users on DECnet/E systems. However, the calls were not designed with COBOL applications in mind. To utilize the full power of the extended interface, you need to access one-byte binary items and set bits -- a difficult task with COBOL -- and even code MACRO-11 (assembly language) subprograms to execute RSTS/E monitor directives.

Data is sent and received in units called segments. The maximum segment size is determined by the communications hardware, with the largest possible segment being controlled by the network management parameter SEGMENT BUFFER SIZE. The flow control options regulate flow in terms of segments or logical messages (one or more segments, as defined by a bit in the send call argument).

In any case, the following fifteen calls are available. The following discussion gives you some idea of the scope of network message services available with the extended COBOL interface. See Chapter 9 for a full discussion of the interface.

1. **Declare Receiver Call: MDCL** -- The MDCL call lets a network program register for network operations. The program identifies itself by object type code, name, or both. It defines the maximum number of logical links it handles, the maximum number of messages to be queued for it in system buffer space, and a limit on system buffer space to be used on its behalf.

2. **Remove Receiver Call: MREM** -- The MREM call terminates message operations, destroying any logical links still active.

3. **Get Local Node Parameters Call: NTLN** -- The NTLN call returns information to the calling program concerning the local node's network parameters.

4. **Log User Event Call: NTEV** -- The NTEV call permits a user-written program to queue an event to the system event processor for logging.

5. **Send Local Data Message: MSLD** -- Programs using the extended COBOL interface can send up to 532 bytes of data to a program at the local node with this call. It is not necessary to establish a logical link in this case.

6. **Send Local Data Message with Privilege Mask: MSPR** -- Programs using the extended COBOL interface can send up to 524 bytes of data to a program at the local node with this call. It is not necessary to establish a logical link in this case.

7. **Send Connect Initiate Message: NTCI** -- The NTCI call requests a logical link with another program in the network. A 172-byte Connect Data Block is used to address the remote program. It includes numeric values stored as one-byte binary items. In addition, the program specifies a link identifier (it can have more than one link), the flow control option desired, and a maximum segment size for receiving data.

8. **Send Connect Confirm Message: NTCC** -- The NTCC call accepts a request for a logical link connection. As with the NTCI call, the program specifies a link identifier, the type of flow control, and a maximum segment size for receiving data.

9. **Send Connect Reject Message: NTCR** -- The NTCR call rejects a request for a logical link connection.

10. **Send Network Data Message: NTDM** -- The NTDM call transmits user data to a remote program over an established logical link. The amount of data sent is limited to the maximum the remote program wants to receive. Sending a Network Data Message is subject to flow control imposed by the local system, the remote system, and the remote program. See Chapter 4 for a full discussion of flow control.

11. **Send Interrupt Message: NTIN** -- The NTIN call transmits high-priority data to a remote program over an established logical link. All DECnet systems maintain separate data streams for Interrupt and Data Messages. The Interrupt Message is delivered to the remote program ahead of all Data Messages waiting to be processed at the remote end.

12. **Send Link Service Message: NTLS** -- The NTLS call requests data over a logical link when the program has requested either segment or logical message flow control. See Chapter 4 for a full discussion of flow control.

13. **Send Disconnect Message: NTDI** -- The NTDI call breaks an established logical link provided that all Network Data, Interrupt, and Link Service Messages previously sent over the link have been acknowledged by the remote ECL. If they have not been acknowledged, the call terminates with an error and the link is not broken.

14. **Send Link Abort Message: NTLA** -- The NTLA call breaks a logical link immediately. No attempt is made to make sure that messages previously sent over the link have been acknowledged by the remote system.

15. **Receive Call: MRCV** -- The MRCV call retrieves a pending message from system buffer space and delivers it to an area defined within the user program. Any of nine message types can be received, corresponding to the nine message types in items 5 - 13.

Comparison of the Interfaces

In summary, the concise COBOL interface is much simpler to use than the extended COBOL interface. With the concise interface you can:

- Establish one logical link with another program in the network
- Send/Receive data over the link
- Disconnect the link

However, if you want to take advantage of the full power of DECnet/E message services, the extended COBOL interface is there for you to use. The extra features available with the extended interface let you:

- Establish multiple logical links
- Send Interrupt Messages or Local Data Messages
- Control the amount of data sent to your program

Chapter 8

Concise COBOL Interface Calls

This chapter presents the specific calls, with detailed formats, used to access the DECnet/E message services with the concise COBOL interface.

Programming Background

To use the DECnet/E message services available with the concise COBOL interface, you code subprogram calls with the CALL statement, including the USING phrase to pass operand values to the subprogram. These subprogram calls reference MACRO-11 subprograms stored in the system library that must be linked with your program. Thus, the process involves coding, compiling, and linking. The following sections briefly describe these aspects of using the DECnet/E concise COBOL interface. Specific call formats are given in the sections "Connect Request Call: CNTCON" through "Disconnect Call: CNTDIS."

Coding

The general format of the concise COBOL message calls is as follows:

```
CALL "name" USING arg1, arg2, ....
```

where:

name is the name of one of the five subprograms available for message services. Each subprogram processes a call at execution time by executing the following steps:

1. Checking for the proper number of arguments in the call.
2. Translating the call and arguments into the form recognizable to the RSTS/E monitor.

3. Passing control to the RSTS/E monitor and, thus, to DECnet/E.

The End Communications Layer (ECL) processes the request and returns control to the user program. The status of each call (whether an error occurred) is returned as a COMP value to a data item defined as an argument in each call.

arg... is the argument list for the particular call. You must define each argument as a data item in the Data Division of the program.

Note that the message subprograms are MACRO-11 (assembly language) object code subprograms and the numeric arguments passed and returned require definition in the Data Division of your COBOL program such that they fit in one computer word. You can define such a COMP data item with PICTURE statements. For example:

PIC 9(1) COMP through PIC 9(4) COMP (for unsigned values)

or

PIC S9(1) COMP through PIC S9(4) COMP (for signed values)

When a COMP data item is defined as PIC S9(4), for example, values ranging from -32768 through 32767 can be returned to the data item (even though 5 character positions are used). To pass a value of more than four characters, you MOVE (in the Procedure Division) the value to the data item used as an argument in the call, rather than use the VALUE IS clause in the Data Division.

Compiling and Linking

The DECnet/E subprograms are stored as object modules in the RSTS/E system library (LB:SYSLIB.OLB). References to the subprograms are resolved automatically during the task building process unless you give specific directions otherwise. See the *PDP-11 COBOL User's Guide* for the normal procedure for compiling, merging ODLs, and task building.

Connect Request Call: CNTCON

The CNTCON call requests a logical link with another program in the network. The other program is identified, and the calling program's access rights to the remote system's or program's services are specified with a project-programmer number (PPN), a password, and an account.

COBOL Call Format

```
CALL "CNTCON" USING stat,rnode,rname,ppn,pwd,act,reason
```

Argument Descriptions

stat

The *stat* argument is a one-word COMP data item (see the section "Coding"). It is set to zero if the CNTCON call succeeds. If an error occurs, *stat* is set to one of the error codes listed under "Returned Status" at the end of this section.

rnode

The *rnode* argument defines the remote node to which the connect request is directed. It consists of one to six uppercase, alphanumeric characters, left-justified in a 6-byte field and (if necessary) padded to six characters with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically. Leading or embedded spaces are invalid. If no node name is to be specified (that is,if the connect request is directed to a program at the local node), the *rnode* argument should contain six spaces.

rname

The *rname* argument is a 21-character alphanumeric data item identifying the remote program with which the connection is to be made. Two types of identification are allowed: object type code or name. The call must specify whichever identifier is used by the remote program. See the section "Network Addressing" in Chapter 4 for a discussion of those network addressing features that are of interest to concise COBOL interface users.

The *rname* argument can take one of the following two formats:

TASK=*name*

or

objtyp=

where:

name Is the alphanumeric name used by the remote program to identify itself as a DECnet program. It can be from 1 to 16 characters long. Note that names on DECnet/E systems can be from one to six uppercase, alphanumeric characters or the special characters "$" (dollar sign), "." (period), and "_" (underscore). Embedded or leading spaces are invalid.

objtyp Is the numeric object type code (in the range from 1 to 255) used by the remote program to identify itself as a DECnet program. The numeric characters must be followed by an equal sign (=) character.

In either format, the characters used must be left-justified in the 21-character data item, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

ppn

The *ppn* argument is a 16-character alphanumeric data item. It is a PPN (referred to on some systems as a user identification code, or UIC). The *ppn* is one of three arguments that you can use to define the calling program's access rights to the remote system's or remote program's services. Normally, a receiving DECnet/E system passes the information on to the receiving program. However, DECnet/E programs using the concise COBOL interface can specify that a corresponding *ppn* argument be verified against that of the incoming request.

The PPN must be left-justified in the 16-character field, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

pwd

The *pwd* argument is an 8-character alphanumeric data item. It is a password -- one of three arguments that define this program's rights to the remote system's or the remote program's services. Normally, receiving DECnet/E systems pass the information on to the receiving program. However, DECnet/E programs using the concise COBOL interface can specify that a corresponding *pwd* argument be verified against that of the incoming request.

The password must be left-justified in the 8-character data item, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

act

The *act* argument is a 16-character alphanumeric data item. It is an account number -- one of three arguments defining this program's access rights to the remote system's or the remote program's services. Normally, receiving DECnet/E systems pass the information on to the receiving program. However, DECnet/E programs using the concise COBOL interface can specify that a corresponding *pwd* argument be verified against that of the incoming request.

The account number must be left-justified in the data item, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

reason

The *reason* argument is a one-word COMP data item. It is set to a reason code that further defines the reason for a rejected link request when the *stat* argument is 5 or 6. See Appendix B for a list of possible reason codes.

Returned Status

stat	Meaning
0	The CNTCON call completed successfully.
1	The program already has an existing logical link. Only one link at a time can be used with the concise COBOL interface. The current link must be disconnected before a CNTCON call can succeed.
2	The format of the call is incorrect. For example, you specified the wrong number of arguments in the call or their lengths are not in the proper range.
3	The CNTCON call was issued by a program without the necessary privileges. A RSTS/E program must have GACNT or WACNT privilege when it executes a CNTCON call.
4	The node specified is unknown or is currently inaccessible (the node is shut down, the line is down, and so forth). Or, the node name is invalid.
5	The CNTCON call was rejected by the remote system or the remote program; or five minutes has elapsed with no response from the remote program.

Returned Status (Cont.)

stat	Meaning
6	The physical line to the remote node went down before the remote program responded to the connect request.
14	The CNTCON call failed for some reason that requires an operator to fix. For example, DECnet/E has not been enabled at the local node; or the program (job) has been removed as a receiver (denied the use of DECnet) by another user on the system.
16	The format of the *rname* argument was invalid.

Example

The following example shows a request for a logical link connection with a remote program named INVTRY at node BOSTON. A PPN of 1,222 and a password of MYNAME are used. The account is 3TF. For example:

```
(Data Division)

01      STAT PIC 9(4) COMP.
01      TASK-NAME PIC  X(21) VALUE IS "TASK=INVTRY".
01      NODE-NAME PIC X(16) VALUE IS "BOSTON".
01      PPN PIC X(16) VALUE IS "1,222".
01      PASWRD PIC X(8)  VALUE IS "MYNAME".
01      ACCT PIC X(16) VALUE IS "3TF".
01      RSN PIC 9(4) COMP.
          .
          .
          .

(Procedure Division)

        CALL "CNTCON" USING STAT, NODE-NAME, TASK-NAME, PPN,
         PASWRD, ACCT, RSN.
```

Connect Accept Call: CNTACP

The CNTACP call accepts a logical link requested by another program in the network. The program issuing the CNTACP call identifies itself either by object type code, name, or both. The project-programmer number (PPN), password, and account number arguments specified in the CNTACP call are checked against corresponding values sent by the remote program in the incoming connect request. If they do not agree, the connection is rejected and the CNTACP call terminates with a status value of 5 in the *stat* argument.

If no request for a logical link has been received when the CNTACP call is issued, the program waits until a connection is requested. Thus, a program can be designed to be run manually with the RUN command rather than by automatic startup. Such a program would issue a CNTACP call to identify itself and then wait until a connect request is received from the remote program.

COBOL Call Format

```
CALL "CNTACP" USING stat,name,objtyp,ppn,pwd,act
```

Argument Descriptions

stat

The *stat* argument is a one-word COMP data item (see the section "Coding"). It is set to zero if the CNTACP call succeeds. If an error occurs, *stat* is set to one of the error codes listed under "Returned Status" at the end of this section.

name

The *name* argument is a 6-character alphanumeric data item. It identifies the calling program in the network environment and it must be unique to the node. If the remote program addresses this one by object type code and this program is not to be identified by name, this argument can be padded with all spaces.

In any case, the name consists of 1 to 6 uppercase, alphanumeric characters or the special characters "$" (dollar sign), "." (period), and "_" (underscore). Leading or embedded spaces are invalid.

The name must be left-justified in the 6-character field, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

objtyp

The object type code (*objtyp*) argument is a 3-character alphanumeric data item expressing an object type code within the range of 0 to 255. If nonzero, it identifies the calling program when the remote program's connect request addressed this program by object type code.

If the remote program addresses this one by name, the *objtyp* argument can be an alphanumeric 0 character. This indicates that object type code addressing is not used. Only one copy of the program can be executed at a time in this case.

The object type code must be left-justified in the data item, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

ppn

The *ppn* argument is a 16-character alphanumeric data item. It is a PPN (referred to on some systems as a user identification code, or UIC). The *ppn* is one of three arguments that, if specified, is checked against the corresponding value in the incoming connect request. If all three arguments match, the call succeeds. If the *ppn* value specified in the CNTACP call does not match the *ppn* value in the incoming connect request, the connection is rejected and the CNTACP call terminates with a status value of 5 in the *stat* argument.

The PPN must be left-justified in the 16-character field, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

pwd

The *pwd* argument is an 8-character alphanumeric data item. It is a password -- one of three arguments checked against corresponding values in the incoming connect request. If all three arguments match, the call succeeds. If the *pwd* argument does not match the *pwd* argument in the incoming connect request, the connection is rejected and the CNTACP call terminates with a status value of 5 in the *stat* argument.

The password must be left-justified in the 8-character field, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

act

The *act* argument is a 16-character alphanumeric data item. It is an account number -- one of three arguments checked against corresponding values in the incoming connect request. If all three arguments match, the call succeeds. If the *act* argument does not match the *act* value

in the incoming connect request, the connection is rejected and the CNTACP call terminates with a status value of 5 in the *stat* argument.

The account number must be left-justified in the 16-character field, and (if necessary) padded with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically.

Returned Status

stat	Meaning
0	The CNTACP call completed successfully.
1	The program already has an existing logical link. Only one link at a time can be used with the concise COBOL interface. The current link must be disconnected before a CNTACP call can succeed.
2	The format of the call is incorrect. For example, you specified the wrong number of arguments in the call or their lengths are not in the proper range.
3	The CNTACP call was issued by a program without the necessary privileges. A RSTS/E program must have SYSIO privilege when it executes a CNTACP call.
5	The CNTACP call was rejected because the *ppn*, *pwd*, or *act* values in the incoming connect request did not match the values of the corresponding arguments in the CNTACP call.
6	The physical path to the remote node went down while the connection sequence was in progress.
13	The alphanumeric *name* used in the *rname* argument is already in use on the local system. Only one program at a time can use a particular name on the local system (see the section "Network Addressing" in Chapter 7).
14	The CNTACP call failed for some reason that requires an operator to fix. For example, DECnet/E has not been enabled at the local node; or the program (job) has been removed as a receiver (denied the use of DECnet) by another user on the system.

Example

The following example accepts a connect request. Note that it would accept the connection requested by the example shown for a CNTCON call in the section "Connect Request Call: CNTCON". For example:

(Data Division)

```
01      STAT PIC 9(4) COMP.
01      NET-NAME PIC X(6)  VALUE IS "INVTRY".
01      OBJTYP PIC X(3)  VALUE IS "0".
01      PPN PIC X(16) VALUE IS "1,222".
01      PSWD PIC X(8) VALUE IS "MYNAME".
01      ACCOUNT PIC X(16) VALUE IS "3TF".
          .
          .
          .
```

(Procedure Division)

```
        CALL "CNTACP" USING STAT, NET-NAME, OBJTYP, PPN,
         PSWD, ACCOUNT.
```

Send Record Call: CNTSND

The CNTSND call sends up to 32767 bytes of data to a remote program over an established logical link. Control returns to the program when the send is complete (that is, all data has been sent and received by the remote program) or when the logical link is disconnected.

Because a CNTSND call lets you send a large amount of data, its use requires some care. For example, on a RSTS/E system network data coming over a logical link is stored in system buffer space until the program receives the data into its own buffer space. If the remote program never receives the data, and there is not enough system buffer space available to hold the entire message, the remote system inhibits transmission by the local program over the link until the congestion at the remote system is alleviated. This condition is known as backpressure (see the section "Backpressure Flow Control" in Chapter 4). Because backpressure is a dynamic network condition, the subprogram that processes the CNTSND call is designed to continually attempt to send the entire message. If the remote program cannot receive the data being sent, the local CNTSND subprogram continues trying to send indefinitely.

One reason the remote program might not receive the data is that it is also using the CNTSND call to transmit a large message while the local system has inhibited incoming message flow on its end of the link. Since neither CNTSND call can terminate until the entire message is sent, both programs wait indefinitely and a deadlock condition results.

Thus, when two programs communicate using the concise COBOL interface, one should be a sender and the other a receiver, or they should establish some protocol of their own for timing send and receive requests.

COBOL Call Format

```
CALL "CNTSND" USING stat,buffer,buflen
```

Argument Descriptions

stat

The *stat* argument is a one-word COMP data item (see the section "Coding"). It is set to zero if the CNTSND call succeeds. If an error occurs, *stat* is set to one of the error codes listed under "Returned Status" at the end of this section.

buffer

The *buffer* argument is a data-name referring to the first character position (byte) of a buffer (record) from which the data is to be sent. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name.

buflen

This argument is a one-word COMP data item defining the number of bytes in the message. The value of this argument must take into account how the message data in the buffer has been defined. See the *PDP-11 COBOL Language Reference Manual* for a description of the USAGE clause to see how much computer storage COBOL data items take. Acceptable values for *buflen* range from 0 through 32767.

Returned Status

stat	Meaning
0	The CNTSND call completed successfully.
1	The program has not established a logical link. A CNTCON or CNTACP call must be executed before the CNTSND call can succeed.
2	The format of the call is incorrect. For example, you specified the wrong number of arguments.
7	The logical link was disconnected before the complete message was sent. This could occur because the remote program disconnected the link or because of some network or system malfunction (the physical path went down, the remote system crashed, and so forth).
14	The CNTSND call failed for some reason that requires an operator to fix. For example, DECnet/E has not been disabled at the local node; or the program (job) has been removed as a receiver (denied the use of DECnet) by another user on the system.

Example

(Data Division)

```
01      PAYREC.
        02 EACH-ONE OCCURS 20 TIMES.
         03 NAME.
          04 LAST-NAME PIC X(20).
          04 FIRS-NAME PIC X(20).
          04 MID-INIT PIC X(1).
         03 HOURS PIC X(3).
         03 HOURLY-WAGE PIC X(4).
01      STAT PIC 9(4).
01      BUFLEN PIC 9(4) COMP VALUE IS 960.
          .
          .
          .
```

(Procedure Division)

```
        CALL "CNTSND" USING STAT, PAYREC, BUFLEN.
```

Receive Record Call: CNTRCV

The CNTRCV call receives a message from a remote program. The call defines a buffer area where the message is to be stored, the size of the buffer in bytes, and a COMP data item that is set to the number of bytes in the received message. Control returns to the program when the receive is complete or the logical link is disconnected.

Note that it is possible to suspend program execution indefinitely on a CNTRCV call if the remote program never sends a message to be received and the logical link is never disconnected. As discussed with the CNTSND call, two programs using the concise COBOL interface should be designed so one is always a sender and the other a receiver, or they should establish their own protocol for send and receive requests.

COBOL Call Format

```
CALL "CNTRCV" USING stat,buffer,buflen,datlen
```

Argument Descriptions

stat

The *stat* argument is a one-word COMP data item (see the section "Coding"). It is set to zero if the CNTRCV call succeeds. If an error occurs, *stat* is set to one of the error codes listed under "Returned Status" at the end of this section.

buffer

This argument is a data-name referring to the first character position (byte) of a buffer to contain the received message. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in fields referenced by data-name. The format must reflect the type of data sent from the remote program.

buflen

The *buflen* argument is a one-word COMP data item defining the length of the buffer in bytes. Acceptable values range from 0 through 32767. If the received message contains more bytes than indicated by the *buflen* argument, the remainder of the message is discarded.

datlen

The *datlen* argument is a one-word COMP data item. When the call completes, it contains the actual number of bytes transferred to the area defined by the *buffer* argument.

Returned Status

stat	Meaning
0	The CNTRCV call completed successfully.
1	The program has not established a logical link. A CNTCON or CNTACP call must be executed before the CNTRCV call can succeed.
2	The format of the call is incorrect. For example, you specified the wrong number of arguments in the call.
7	The logical link was disconnected before the entire message was received by the local ECL. Some DECnet user interfaces (including the DECnet/E extended COBOL interface) allow a program to abort a logical link; that is, to disconnect the logical link regardless of whether an entire message has been transferred. This may have happened; or the link may have been disconnected because the line went down or the remote system crashed.
8	The logical link no longer exists because the remote program has disconnected the link. This is regarded as an EOF for concise COBOL programs.
9	The length of the message received exceeded the number of bytes defined by the *buflen* argument. The remaining data was discarded.
14	The CNTRCV call failed for some reason that requires an operator to fix. For example, DECnet/E is disabled at the local node; or the program (job) has been removed as a receiver (denied the use of DECnet) by another user on the system.

Example

The following example receives up to 1000 bytes of data from the network over an established logical link, storing it in a data record named INBUF. For example:

```
(Data Division)

01      INBUF PIC X(1000).
01      STAT PIC 9(4) COMP.
01      BUFLEN PIC 9(4) COMP VALUE IS 1000.
01      DATLEN PIC 9(4) COMP.
          .
          .
          .

(Procedure Division)

        CALL "CNTRCV" USING STAT, INBUF, BUFLEN, DATLEN.
```

Disconnect Call: CNTDIS

The CNTDIS call disconnects a logical link. If the remote program uses the DECnet/E concise COBOL interface and is executing a CNTRCV call, that CNTRCV terminates with the end-of-file (EOF) status value (*stat* = 8).

COBOL Call Format

```
CALL "CNTDIS" USING stat
```

Argument Descriptions

stat

The *stat* argument must be a one-word COMP data item. It is set to zero if the CNTDIS call succeeds. If an error occurs, *stat* is set to one of the error codes listed under "Returned Status."

Returned Status

stat	Meaning
0	The CNTDIS call completed successfully.
1	The program has not established a logical link. A CNTCON or CNTACP call must be executed before the CNTDIS call can succeed.
2	The format of the call is incorrect. For example, you specified the wrong number of arguments in the call.
10	There are still messages waiting to be received with a CNTRCV call. A program cannot disconnect the link until it has processed all the messages that have been sent to it.
11	The link has already been disconnected.
14	The CNTDIS call failed for some reason that requires an operator to fix. For example, DECnet/E is disabled at the local node; or the program (job) has been removed as a receiver (denied the use of DECnet) by another user on the system.

Example

The following example disconnects the program's logical link:

```
(Data Division)

01      STAT PIC 9(4) COMP.
          .
          .
          .

(Procedure Division)

        CALL "CNTDIS" USING STAT.
```

Chapter 9

Extended COBOL Interface Call Formats

This chapter presents the specific calls, with detailed formats, used to access the DECnet/E message services with the extended COBOL interface.

Programming Background

To use the RSTS/E message services in the extended COBOL interface, you code subprogram calls with the CALL statement, including the USING clause to pass operands to the subprograms. These subprograms reference MACRO-11 subprograms stored in the system library that must be linked with your program. Thus, the process involves coding, compiling, and linking. The following sections briefly describe these aspects of using the extended DECnet/E interface in COBOL. Specific call formats are given in the sections "Declare Receiver: MDCL" through "Receive: MRCV."

Coding

The general format for the extended COBOL message calls is:

```
CALL "name" USING arg1, arg2,...
```

where:

name is the name of one of the 14 subprograms available for message services. Each subprogram processes a call at execution time by executing the following steps:

1. Checking for the proper number of arguments in the call.
2. Translating the call and arguments into the form recognizable to the RSTS/E monitor.

3. Passing control to the RSTS/E monitor and, therefore, to DECnet/E.

DECnet/E processes the request and returns control to the user program. If an error occurs, DECnet/E returns an error value to an integer variable defined as an argument in each call.

arg... is the argument list for the particular call. You must define each argument as a data item in the Data Division of the program.

Note that the message subprograms are MACRO-11 (assembly language) object code subprograms and the numeric arguments passed and returned require definition in the Data Division of your COBOL program so they fit in one computer word. You can define a one-word COMP data item with PICTURE statements. For example:

PIC 9(1) COMP through PIC 9(4) COMP (for unsigned values)

or

PIC S9(1) COMP through PIC S9(4) COMP (for signed values)

Once you define a COMP data item as PIC S9(4), for example, values ranging from -32768 through 32767 can be returned to the data item (even though five character positions are used). To pass a value of more than 4 characters, you MOVE (in the Procedure Division) the value to the data item, rather than use the VALUE IS clause in the Data Division.

Also, with the extended COBOL interface, you must be able to access one-byte binary values. Some general tips on how to do this are given below. Other examples are given in the call format discussions in the sections "Declare Receiver: MDCL" through "Receive: MRCV."

Tips for Returned Status Information in Byte Form

The Receive call and three send calls (Send Network Data, Interrupt, and Link Service Messages) return status and control information to a 40-byte user buffer. Some of this information consists of signed and unsigned numeric byte values. PDP-11 COBOL does not recognize such a data category. Thus, to use this information you must convert them to one-word integer values. The technique is different for signed and unsigned byte values. Two examples follow. The first converts signed one-byte binary items to signed one-word COMP items. If the value expected is within the range -128 through 127, you can use the technique shown in the first example. The second example converts unsigned one-byte binary items (ranging from 0 to 255) to unsigned

one-word COMP items. If the value expected is within the range 0 to 255, use this or a similar technique:

Example 1 -- Signed

```
IDENTIFICATION DIVISION.
PROGRAM ID.  BYTSGN.
AUTHOR.
        T-BOBFORD.
DATE-WRITTEN.  24-SEP-86.
DATE-COMPILED.
        24-SEP-86.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.

DATA DIVISION.
WORKING-STORAGE SECTION.
01      WORK-WORD PIC S9(4) COMP.
01      WW REDEFINES WORK-WORD.
        03  LOW-BYTE PIC X.
        03  HIGH-BYTE PIC X.
01      SEVEN-BITS-G PIC 9(4) COMP VALUE 127.
01      SEVEN-BITS REDEFINES SEVEN-BITS-G PIC X.
01      ALL-BITS-G PIC 9(4) COMP VALUE 255.
01      ALL-BITS REDEFINES ALL-BITS-G PIC X.
LINKAGE SECTION.
01      ONE-BYTE   PIC X.
01      ONE-WORD   PIC S9(4) COMP.

PROCEDURE DIVISION USING ONE-BYTE ONE-WORD.
SBEGIN.
        IF ONE-BYTE > SEVEN-BITS
          MOVE ALL-BITS TO HIGH-BYTE
        ELSE
          MOVE 0 TO WORK-WORD.
        MOVE ONE-BYTE TO LOW-BYTE.
        MOVE WORK-WORD TO ONE-WORD.
        EXIT PROGRAM.
```

Example 2 -- Unsigned

```
  IDENTIFICATION DIVISION
  PROGRAM-ID.  BYTUNS.
  AUTHOR.
          T-BOBFORD.
          DATE-WRITTEN.  24-SEP-86.
          DATE-COMPILED.
          24-SEP-1986.

  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. PDP-11.
  OBJECT-COMPUTER. PDP-11.

  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01      WORK-WORD PIC 9(4) COMP.
  01      WW REDEFINES WORK-WORD.
          03  LOW-BYTE PIC X.
          03  HIGH-BYTE PIC X.
  LINKAGE SECTION.
  01      ONE-BYTE PIC X.
  01      ONE-WORD PIC S9(4) COMP.

  PROCEDURE DIVISION USING ONE-BYTE ONE-WORD.
  SBEGIN.
          MOVE 0 TO WORK-WORD.
          MOVE ONE-BYTE TO LOW-BYTE.
          MOVE WORK-WORD TO ONE-WORD.
          EXIT PROGRAM.
```

Note that not all the status information returned needs to be used in this way. Some values are returned as signed or unsigned one-word COMP data items that you can reference directly with data-names defined as PIC S9(4) COMP or PIC 9(4) COMP.

Tips for Setting Bits

Some of the argument descriptions in the extended COBOL interface calls require that you set bits in one-word COMP data items. Such arguments are usually described as the sum of the bit values, for example:

```
Y = X1 + X2 + ...  + Xi + ...  Xn.
```

The Xi components are then described in terms of what happens when Xi equals zero or some power of two: 2, 4, 8, ..., 128. Setting a

component to the nonzero value sets the bit that has that value in the computer word. Setting a zero value clears the bit.

Tips for Testing Bits

As mentioned previously, the Receive call and three send calls return status information to a 40-byte user buffer. In some cases, you may want to test bits in one-byte binary items. A sample technique for COBOL programs follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
        BITS.
AUTHOR.
        T-BOBFORD.
INSTALLATION.   DIGITAL EQUIPMENT CORPORATION.
DATE-WRITTEN.
DATE-COMPILED.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.

DATA DIVISION.
WORKING-STORAGE SECTION.
01      BITS-TESTER PIC 9(4) COMP VALUE 0.
01      TEST-BYTE REDEFINES BITS-TESTER PIC X.
01      BIT-WORK PIC S9(4) COMP.
01      BIT-VALUE PIC 9(4) COMP.
01      BIT-SUB PIC 9(4) COMP.
01      THE-BITS.
        03 0-BIT PIC X.
        03 1-BIT PIC X.
        03 2-BIT PIC X.
        03 3-BIT PIC X.
        03 4-BIT PIC X.
        03 5-BIT PIC X.
        03 6-BIT PIC X.
        03 7-BIT PIC X.
01      BIT-ARRAY REDEFINES THE-BITS.
        03 A-BIT PIC X OCCURS 8.
01      BYTE-YOU-WANT-TO-TEST PIC X.

PROCEDURE DIVISION.
TESTER SECTION.
SBEGIN.
        MOVE BYTE-YOU-WANT-TO-TEST TO TEST-BYTE.
        PERFORM COMPUTE-BITS.
```

Then, for example, to test that the two low-order bits are set:

```
IF 0-BIT = "1" AND 1-BIT = "1" ...

COMPUTE-BITS SECTION.
STARTIT.
        MOVE ALL "0" TO THE-BITS.
        MOVE 128 TO BIT-VALUE.
        MOVE 8 TO BIT-SUB.
LOOP.
        PERFORM COMP-BITS.
        SUBTRACT 1 FROM BIT-SUB.
        IF BIT-SUB = 0 OR BIT-TESTER = 0
          GO TO GET-OUT
        GO TO LOOP
COMP-BITS.
        SUBTRACT BIT-VALUE FROM BITS-TESTER
          GIVING BIT-WORK.
        IF BIT-WORK NOT NEGATIVE
          MOVE "1" TO A-BIT (BIT-SUB)
          MOVE BIT-WORK TO BITS-TESTER.
        DIVIDE 2 INTO BIT-VALUE.
GET-OUT.
        EXIT.
```

Compiling and Linking

The DECnet/E subprograms are stored as object modules in the RSTS/E system library (LB:SYSLIB.OLB). The Task Builder (TKB) automatically resolves the references to the message subroutines from the system library, unless you give specific directions. Follow your normal procedure for compiling, merging ODLs, and task building. See the *PDP-11 COBOL User's Guide* for a description.

Declare Receiver: MDCL (Local and DECnet)

You must execute the MDCL call before a program can receive any messages or send any network messages. (Local messages can be sent without first executing this call.) The call defines an identifying object type code or name (or both) and any restrictions on incoming messages. The monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block (RIB) for the job (see the section "DECnet/E Use of the Declare Receiver Call" in Chapter 3). It remains in effect until a Remove Receiver call (MREM) is executed for the job.

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 RIBs, numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call. The system allocates space for each RIB from the small buffer pool.

Multiple RIBs can be useful in a program that performs several functions. For example, in certain applications, you might want to use one RIB for high priority messages or messages from specially privileged senders, and another for lower priority messages or messages from any sender. If the program consists of several subroutines or modules, each one can process messages independently using different RIBs.

COBOL Call Format

```
CALL "MDCL" USING errval,name,objtyp,access,zero,mmax, lmax,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the MDCL call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

name

The logical name is a data-name referring to the first character position of a 6-byte area containing one to six characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name is left-justified in the 6-byte area and (if necessary) padded to six characters with spaces. Note that the VALUE IS clause or MOVE statement does this automatically. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), this area should contain six spaces.

The name identifies the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code.

For local or network programs to access the calling program by logical name, the name must be nonnull and unique to the node. That is, only one program at the node can declare its identity with that name at any given time. The End Communication Layer (ECL) uses the logical name to identify the calling program for queuing messages from local or network programs. If the logical name is null, only network access by object type code is allowed. See the discussion of *name*, *objtyp*, and *access*.

objtyp

This one-word COMP data item specifies the object type code, another form of network addressing. As discussed in the section "Registering with the Monitor: Declare Receiver Call in Chapter 3," the object type defines a service that the calling program performs. If the program issuing the MDCL call is addressed by object type alone (*name* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environment. Unlike the logical name, the object code need not be unique within the node -- multiple copies of a program can declare their identity with the same type code at the same time. If *name* is null, *objtyp* cannot be zero (see the section "Network Addressing" in Chapter 4).

Acceptable values for the object type code range from 0 through 255. The value 0 indicates that other programs in the network never address the calling program by object type code. The range 1 to 127 is reserved for DECnet use (see Appendix A). The range 128 to 255 is available for definition and use by a network installation.

access

This one-word COMP data item contains the sum of five bit values ($l + p + n + o + s$) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values (*l*, *p*, and *n*) limit the types of senders that can queue messages for the calling program. However, they do not limit the messages that the program can send.

If *l* = 0, messages from local senders are not queued for the calling program. If *l* = 1, messages from local senders are queued. Note that local senders who use the network functions are considered network senders in this context.

If *p* = 0, incoming messages from local senders with and without privilege are queued for the calling program. If *p* = 2, incoming messages from local senders are queued only if the sender has SEND privilege. Note that this bit is ignored if *l* = 0, that is, if no local senders are allowed.

If *n* = 0, Session Control rejects incoming requests for logical links. If *n* = 4, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access.

Table 9-1 summarizes access code values and the access permitted for each.

Table 9-1: Types of Access Permitted

Access Code (l+p+n)	Network Logical Links	Local Senders with Privilege	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit value *o* is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If *o* = 0, incoming requests for logical links do not affect the link maximum. However, if *o* = 8 after queuing an incoming Connect Initiate Message to the receiver, ECL sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 8, a program prevents the possibility of a Connect Initiate Message being queued to it after its one logical link is disconnected and before it issues a Remove Receiver call. For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote system, Session Control queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that the first link is broken and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value *s* is used to modify the function of the RSTS/E conditional Sleep monitor call. If *s* = 0, any unreceived messages queued for the program blocks the execution of a conditional Sleep call. This is the normal RSTS/E function. However, if *s* = 16 the RSTS/E monitor does not check the program's message queue when determining whether or not it should suspend program execution. Several other conditions (such as receiving a delimiter on an open terminal) can still block the sleep, but a pending message does not.

name*, *objtyp*, and *access

Session Control checks the *name*, *objtyp*, and *access* arguments to make sure that they are consistent. For example, a null logical name and an access allowing local senders are inconsistent because local senders can address the program by logical name. DECnet/E checks these arguments as shown in Table 9-2. If the arguments are invalid, DECnet/E returns an error code of 18 in the *errval* argument. If the arguments are valid, DECnet/E allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (*errval* = 4). A retry may succeed.

Table 9-2: System Validation of Name, Object, and Access Arguments

Access Code (1+p+n)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local (only)	Ignored	Must be nonnull and unique.
4,6	Network (only)	Zero	Must be nonnull and unique.
		Nonzero	If *name* is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If *name* is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.
5,7	Network or Local	Any value	Must be nonnull and unique so local senders can refer to the calling program by logical name. Network senders can use the logical name or object type (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

zero

Pass a one-word COMP data item of 0 here. This field is reserved for compatibility with previous releases.

mmax

The message maximum (*mmax*), a one-word COMP data item ranging from 0 to 255, limits the number of messages queued for the calling program.

ECL keeps count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the message maximum set by a receiver:

- Incoming Local Data Messages are no longer queued for the receiving program. An error is returned to the local sender.
- Incoming Connect Initiate Messages cause ECL to determine whether another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
- An incoming Network Data Message causes ECL to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message. This forces the remote system to retransmit when flow is reenabled. See the section "Backpressure Flow Control" in Chapter 4 for further details.

lmax

The link maximum (*lmax*), a one-word COMP data item ranging from 0 to 255, limits the number of incoming requests for logical links for the program. The program must have SYSIO privilege for the link maximum to be nonzero.

If the number of links currently active is greater than or equal to *lmax* when a remote Connect Initiate Message is received for the program, the local ECL checks to determine whether another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting *n* = 0 in the *access* argument. That is, Session Control rejects all incoming requests for logical links to the calling program. The *lmax* argument does not, however, limit the number of logical links initiated by the program. That is, it does not stop the program from sending Connect Initiate Messages.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 to 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the last case, issue a Remove Receiver call followed by another Declare Receiver call. (It is common practice to code a Remove Receiver call immediately before a Declare Receiver call.)
4	NO ROOM FOR USER ON DEVICE There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.
10	PROTECTION VIOLATION The calling program must have SYSIO privilege at the time the Declare Receiver call is executed.
16	NAME OR ACCOUNT NOW EXISTS The logical name specified is being used by another job.
18	ILLEGAL SYS() USAGE The parameters specified are inconsistent. See the discussion of *logical name*, *object*, and *access*.

Example

The following example shows a Declare Receiver call allowing both network and local access. Both a logical name (CNET) and an object type code (142) are specified. The buffer maximum is set to 512 and the message maximum is set to 5. No more than 512 bytes of user message data from incoming Local Data Messages are stored in monitor pool space. And, if more than five messages accumulate for the program, an incoming request for a connection causes ECL to check to determine whether another copy of the program can be started automatically. If not, Session Control rejects the link and any incoming data messages causes ECL to inhibit incoming message flow on all the program's logical links because of the backpressure condition. The link maximum is set to 2. When the program has established two logical links, an incoming Connect Initiate Message causes ECL to check to determine whether another copy of the program can be started. If not, Session Control rejects the link. For example:

(Data Division)

```
01      STAT PIC 9(4) COMP.
01      DECLARE-ITEMS.
        02 NET-NAME PIC X(6) DISPLAY VALUE IS "CNET  ".
        02 OBJTYP PIC 9(4) COMP VALUE IS 142.
        02 ACCESSM PIC 9(4) COMP VALUE IS 5.
        02 BMAX PIC 9(4) COMP VALUE IS 512.
        02 MMAX PIC 9(4) COMP VALUE IS 5.
        02 LMAX PIC 9(4) COMP VALUE IS 2.
        02 RIB PIC 9(4) COMP VALUE IS 0.
          .
          .
          .
```

(Procedure Division)

```
        CALL "MDCL" USING STAT, NET-NAME, OBJTYP, ACCESSM,
         BMAX, MMAX, LMAX, RIB
```

Remove Receiver: MREM (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. Session Control discards all pending messages and aborts any active logical links. Execute this call at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Programs with JOBCTL privilege can remove other RSTS/E jobs with this call, although normally only system utility programs do this.

COBOL Call Format

```
CALL "MREM" USING errval,jobx2,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the MREM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

jobx2

This argument is a one-word COMP data item. A zero value for *jobx2* removes the calling job. Programs with JOBCTL privilege can remove another local RSTS/E job by specifying job number times two for this argument.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
10	PROTECTION VIOLATION The caller does not have JOBCTL privilege and attempted to remove another job (that is, *jobx2* is nonzero).
18	ILLEGAL SYS() USAGE The *jobx2* argument was odd. *Jobx2* must be zero to remove the calling program or job number times two to remove another job.

Example

The following example shows a typical Remove Receiver call, removing the calling job from the system's list of declared receivers:

(Data Division)

```
01      ERRVAL PIC  9(4) COMP.
01      JOBX2 PIC 9(4) COMP VALUE IS 0.
01      RIB PIC 9(4) COMP VALUE IS 0.
          .
          .
          .
```

(Procedure Division)

```
        CALL "MREM" USING ERRVAL, JOBX, RIB.
```

Get Local Node Parameters: NTLN (DECnet)

The NTLN call returns information to the calling program concerning the local node's network parameters. Data returned includes the node name or node number, node address, and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. See the section "Automatic Job Start-Up" in Chapter 4 for a discussion of the default user's account.

COBOL Call Format

```
CALL "NTLN" USING errval,buffer,buflen,bufoff
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTLN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

buffer

This argument is a data-name referring to the first character position (byte) of the buffer to receive the local node information. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by the data-name. The returned data can be offset from the beginning of the buffer (see *bufoff*).

When the NTLN call completes successfully, data is returned to the buffer in the following format:

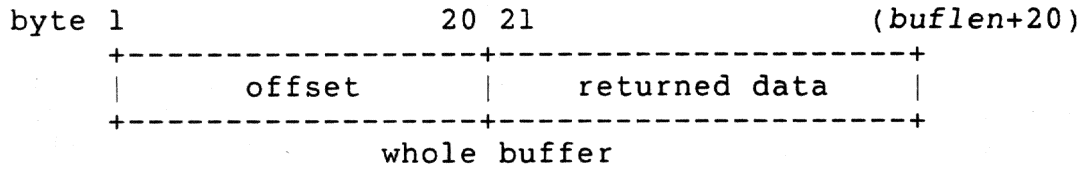
Bytes	Contents
0-5	Local node name, consisting of 1 to 6 uppercase, alphanumeric ASCII characters. The name contains at least one alphabetic character. A name shorter than 6 characters is left-justified and padded to 6 characters with spaces.
6-13	Reserved.
14-15	Local default PPN. Byte 14 = programmer number; byte 15 = project number. If no default account is defined, both bytes are zero.

buflen

This one-word COMP data item defines the number of bytes to be returned to the buffer specified with *buffer*. If *buflen* is less than 16 bytes, the call returns only as much data as there is room for. For example, if *buflen* = 8 bytes, the call returns only the local node name and address. Sixteen bytes are required to receive all the local node parameters.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if offset = 20, the returned data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of 10. If the actual buffer, beginning at *buffer* was only 16 bytes long, the returned data overwrite 10 bytes beyond the end of the buffer.

Possible Errors

errval	Meaning
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTLN call used to obtain all the local node parameters. The node name is to be stored in the data item NODE-NAME; the address, in the COMP data item NODE-ADDRESS; the node number, in the data item NODE-NUMBER; and the default PPN, as a unit in the COMP data item PPN and separately in the COMP data items PROJECT and PROGRAMMER. For example:

(Data Division)

```
01      RETURN-DATA.
        03   NODE-NAME PIC X(6).
        03   RESERVED-BYTES PIC X(8).
        03   PPN PIC 9(4) COMP.
        03   PROJ-PROG REDEFINES PPN.
             05   PROG-BYTE PIC X.
             05   PROJ-BYTE PIC X.
01      BUFFER-LENGTH PIC 9(4) COMP VALUE IS 16.
01      BUFFER-OFFSET PIC 9(4) COMP VALUE IS 0.
01      ERRVAL PIC 9(4) COMP.
01      PROGRAMMER PIC 9(4) COMP.
01      PROJECT PIC 9(4) COMP.
01      WORK-WORD PIC 9(4) COMP.
01      WW REDEFINES WORK-WORD.
        03   LOW-BYTE PIC X.
        03   HIGH-BYTE PIC X.
             .
             .
             .
```

(Procedure Division)

```
        CALL "NTLN" USING ERRVAL, RETURN-DATA, BUFFER-LENGTH,
             BUFFER-OFFSET.
        MOVE 0 TO WORK-WORD.
        MOVE PROG-BYTE TO LOW-BYTE.
        MOVE WORK-WORD TO PROGRAMMER.
        MOVE PROJ-BYTE TO LOW-BYTE.
        MOVE WORK-WORD TO PROJECT.
```

Log User Event: NTEV (DECnet)

The NTEV call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

Note

This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the experienced network user and is not intended for normal network programs.

You need SYSIO and SEND privileges to use NTEV.

COBOL Call Format

```
CALL "NTEV" USING errval,evmod,evcls,evtyp,entyp,enval,
                  buffer,buflen,bufoff
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTEV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

evmod

This one-word COMP data item specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with an error (*errval* = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a Missed event message, and the call terminates with a successful status (*errval* = 0).

Note that in the second case, the user data is not logged. The system only logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet internal events when the event queue is full or there are no more small buffers.

evcls

This one-word COMP data item specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480.

evtyp

This one-word COMP data item specifies the type of event within the class specified by *evcls*. The event type must be in the range of 0 to 31.

entyp

This one-word COMP data item specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

```
-1  No specific entity
 0  Node
 1  Line
 3  Circuit
```

enval

This one-word COMP data item associates a value to the entity specified by the *entyp* argument. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

buffer

This argument is a data-name referring to the first character position of the buffer containing the optional user data to be processed with the event. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data to be processed can be offset from the beginning of the buffer (see *bufoff*).

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. You can use DECnet parameter types if the event concerns a node, a circuit, or a line. In this case, the event logger outputs standard descriptive text for the specific parameter. If you do not use a DECnet parameter type, parameter types in the range of 3900 to 4095 can be specified. See the *DIGITAL Network Architecture Phase IV Network Management*

Functional Specification for further information on event definitions and parameters.

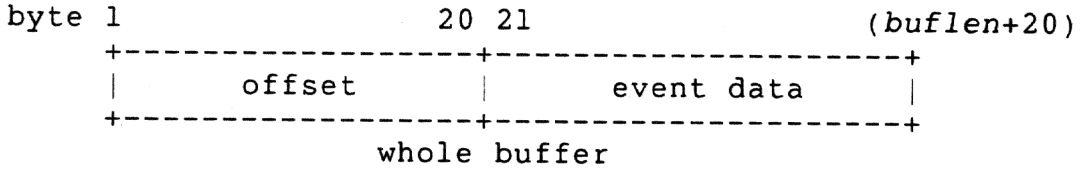
Up to 200 bytes of data can be passed to the event logger for processing.

buflen

This one-word COMP data item defines the number of bytes of data in the buffer specified with *buffer*. Up to 200 bytes of data can be passed. If no data is specified, *buflen* must be 0.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the data to be processed begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *buffer*-100.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod*=0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.

Possible Errors (Cont.)

errval	Meaning
10	PROTECTION VIOLATION The calling program did not have SYSIO or SEND privileges at the time the NTEV call was issued.
18	ILLEGAL SYS() USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet/E was not installed.The network functions cannot be executed.

Example

The following example shows the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of a Local Link Address (*lla*) parameter number (2130) and value. A Missed event message is logged if the event itself cannot be. Note that even though 6 bytes are allocated for the parameter value, only 2 are used. For example:

```
(Data Division)

01  MODIFIER PIC 9(4) COMP VALUE IS 1.
01  EVENT-CLASS PIC 9(4) COMP.
01  EVENT-TYPE PIC 9(4) COMP.
01  ENTITY-TYPE PIC 9(4) COMP.
01  ENTITY-VALUE PIC 9(4) COMP.
01  PARAMETER-DATA.
    03 FILLER PIC X(6).
    03 PARAMETER-NUMBER PIC 9(4) COMP.
    03 DATA-LENGTH PIC X.
    03 DATA-VALUE-1 PIC 9(4) COMP.
    03 DATA-VALUE-2 PIC 9(4) COMP.
    03 DATA-VALUE-3 PIC 9(4) COMP.
01  BUFFER-LENGTH PIC 9(4) COMP.
01  BUFFER-OFFSET PIC 9(4) COMP VALUE IS 6.

01  WORK-WORD PIC 9(4) COMP.
01  WW REDEFINES WORK-WORD.
    03 LOW-BYTE PIC X.
    03 HIGH-BYTE PIC X.
     .
     .
     .
```

(Procedure Division)

```
MOVE 480 TO EVENT-CLASS.
MOVE 3 TO EVENT-TYPE.
MOVE 0 TO ENTITY-TYPE.
MOVE 135 TO ENTITY-VALUE.
MOVE 2130 TO PARAMETER-NUMBER.
MOVE 2 TO WORK-WORD.
MOVE LOW-BYTE TO DATA-LENGTH.
MOVE 5 TO BUFFER-LENGTH.

CALL "NTEV" USING ERRVAL, MODIFIER, EVENT-CLASS,
 EVENT-TYPE, ENTITY-TYPE, ENTITY-VALUE,
 PARAMETER-DATA, BUFFER-LENGTH, BUFFER-OFFSET.
```

The NTEV call in the previous example generates the following event message:

```
Event type 480.3
Occurred 24-Sep-86 15:31:39.7 on node 3.135 (EDERE)
Node address 3.135 (EDERE)
Parameter #2130 = 40960
```

Send Calls

There are ten separate send calls -- two local and eight network. Each of these calls causes ECL to format a message that is then either queued to a local program or transmitted across the network to a remote program.

Send Local Data Message: MSLD (Local)

The MSLD call sends up to 532 bytes of user data to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege or the MSLD call fails with an error (*errval* = 10). The receiving program is identified in the MSLD call either by its job number or by logical name with which it declared itself a receiver.

COBOL Call Format

```
CALL "MSLD" USING errval,job,name,buffer,buflen,bufoff,
                  parptr, parlen
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the MSLD call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

The *job* argument is a one-word COMP data item. If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name

- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

When the *job* argument = 0, this argument is a data-name referring to the first character of a 6-character field containing the logical name identifying the intended local receiver. The receiver must have declared its identity with this name in its Declare Receiver call. Valid characters are uppercase, alphanumerics, and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The characters are left-justified in the field and (if necessary) padded to six characters with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically. Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

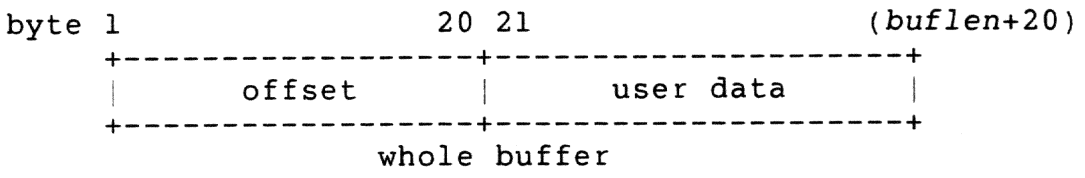
This argument is a data-name referring to the first character position (byte) of a buffer containing the user message data to be sent. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The message data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSLD call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument is a data-name referring to the first character position (byte) of an area containing up to 20 bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the *buffer* area. For example, a RSTS/E COBOL program doing a Receive call that returns a Local Data Message gets the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This one-word COMP data item defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 20.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	ILLEGAL SYS() USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512; the *parlen* argument can range from 0 through 20.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Example

The following example shows a local send to a program that has issued a Declare Receiver call with the logical name LOCAL. A total of 512 bytes are sent from BUFFER. No parameter message data is sent. For example:

```
(Data Division)

01   BUFFER PIC X (512).
01   STAT PIC 9(4) COMP.
01   SEND-LOCAL-ITEMS.
     03   REM-NAME PIC X(6) VALUE IS "LOCAL ".
     03   JOBX2 PIC 9(4) COMP VALUE IS 0.
     03   BUFLEN PIC 9(4) COMP VALUE IS 512.
     03   PARLEN PIC 9(4) COMP VALUE IS 0.
          .
          .
          .

(Procedure Division)

          CALL "MSLD" USING STAT, JOBX2, REM-NAME, BUFFER,
               BUFLEN, BUFFER, PARLEN.
```

Send Local Data Message with Privilege Mask: MSPR (Local)

The MSPR call sends up to 524 bytes of user data, plus a privilege mask supplied by the monitor, to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege or the MSPR call fails with an error (*errval* = 10). The receiving program is identified in the MSPR call either by its job number or by logical name with which it declared itself a receiver.

COBOL Call Format

```
CALL "MSPR" USING errval,job,name,buffer,buflen,bufoff,
                  parptr, parlen
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the MSPR call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

The *job* argument is a one-word COMP data item. If *job*%=0, the call uses the logical name to determine the receiver. If *job*%=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*%=8% directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate)

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver

only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

When the *job* argument = 0, this argument is a data-name referring to the first character of a 6-character field containing the logical name identifying the intended local receiver. The receiver must have declared its identity with this name in its Declare Receiver call. Valid characters are uppercase, alphanumerics, and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The characters are left-justified in the field and (if necessary) padded

to six characters with spaces. Note that the VALUE IS clause or the MOVE statement does this automatically. Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

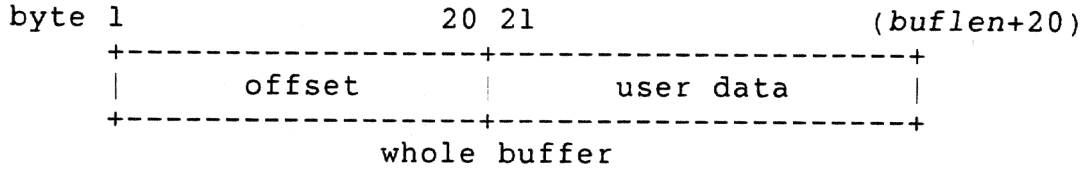
This argument is a data-name referring to the first character position (byte) of a buffer containing the user message data to be sent. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The message data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSPR call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument is a data-name referring to the first character position (byte) of an area containing up to 12 bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the *buffer* area. For example, a RSTS/E COBOL program doing a Receive call that returns a Local Data Message gets the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This one-word COMP data item defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 12.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	ILLEGAL SYS() USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512; the *parlen* argument can range from 0 through 12.
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Send Connect Initiate Message: NTCI (DECnet)

The NTCI call requests a logical link to another program. The call establishes the User Link Address (ULA) that the calling program uses to refer to the link. The other program is identified with a Connect Data Block. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 bytes of user data can be sent with the NTCI call.

COBOL Call Format

```
CALL "NTCI" USING errval,ula,llmod,rmax,buffer,buflen,bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTCI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is a one-word COMP data item in the range of 1 to 255. Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI call is accepted by the remote ECL and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

llmod

The one-word COMP data item *llmod* indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

0 No flow control
1 Segment flow control
2 Message flow control

See Chapter 4 for a full discussion of flow control. Note that each program on a logical link selects its own flow control. The *llmod* value is the selection made by the program issuing the call. If *llmod* is nonzero, the program must issue Network Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space but is not designed to process large messages in small pieces, can impose a limit on the amount of user message data it is willing to receive on a logical link. The *rmax* argument, a one-word COMP data item, specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value. Note that the calling program is informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive: MRCV" for a description of how to process large messages in small pieces.

buffer

This argument is a data-name referring to the first character position (byte) of a buffer containing the Connect Data Block and optional user message data. The 172-byte Connect Data Block and up to 16 bytes of message data are assumed to be in the buffer in contiguous bytes. They can be offset from the beginning of the buffer (see *bufoff*).

The Connect Data Block identifies the target program for the requested logical link. Figure 9-1 shows the format of the Connect Data Block. Table 9-3 describes the fields within the block.

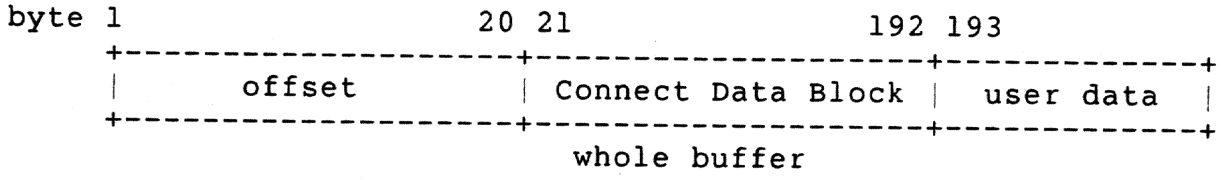
Up to 16 bytes of user data can follow the Connect Data Block. Any PICTURE categories can be used.

buflen

This one-word COMP data item defines the number of bytes to be sent. Since the Connect Data Block is 120 bytes long and the optional user data can be from 0 to 16 bytes long, acceptable values for *buflen* range from 172 through 188.

bufoff

This one-word COMP data item defines an offset, in bytes, from the start of the buffer from which the data is to be sent. For example, if *bufoff* = 20, the Connect Data Block begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. If, for example, a program's buffer is 100 bytes long, with an offset of 20 and a length of 188, 188 bytes are sent; that is, the last 80 bytes of the buffer, and 108 bytes of whatever is beyond the buffer in memory.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

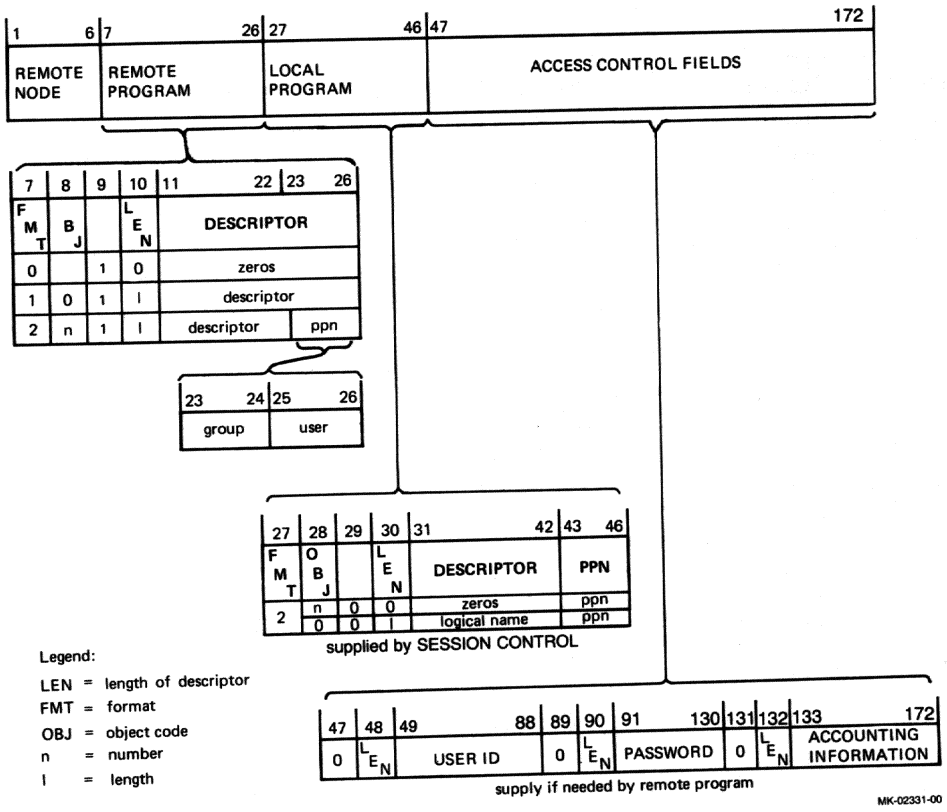


Figure 9-1: Format of Connect Data Block on a Send Connect Initiate Message

Table 9-3: Format of Connect Data Block on a Send Connect Initiate Message

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node to which the Connect Initiate Message is to be sent. Names shorter than 6 characters must be left-justified with space fill to 6 characters. Leading and embedded spaces are invalid. If no node name is specified (that is, the request is directed to a program at the local node), these 6 bytes contain an ASCII 0 and 5 spaces.
7-26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats: **Format 0** The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9	Must be one.
10-26	Zero (Not used).

Table 9-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	Format 1 The remote program is identified by a descriptor. The fields are:

Bytes	Content
7	One, to indicate a format 1 name.
8	Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not).
9	Must be one.
10	Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field.
11-26	Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations.

Format 2

This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

Table 9-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The subfields are: **Bytes** — **Content** 7 — Two, to indicate a format 2 name. 8 — Object type code of the remote program, if applicable. If zero, the descriptor must be used. 9 — Must be one. 10 — Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22). 11-22 — Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field. 23-24 — Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E PPN. 25-26 — Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E PPN.
27-46	**Local Program.** This field is filled in by the local session control using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

Table 9-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with an object code alone, the object code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The RSTS/E PPN under which the program is running is passed in bytes 43-46 as the group and user codes: the project number is in byte 43; the programmer number is in byte 45. Both are in binary format.
47-172	**Access Control Fields.** These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number or system password, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved -- must be zero.
48	Length of user identification information specified in the next field, 0 to 40 bytes.

Table 9-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content	
	49-88	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, set the length (byte 48) to zero.
	89	Reserved -- must be zero.
	90	Length of password information specified in the next field, 0 to 40 bytes.
	91-130	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, set the length (byte 90) to zero.
	131	Reserved -- must be zero.
	132	Length of the accounting information specified in the following field, 0 to 40 bytes.
	133-172	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE

The calling program already has a logical link with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.

6	NOT A VALID DEVICE

The node named in the Connect Data Block is not known to the system.

14	DEVICE HUNG OR WRITE LOCKED

The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.

17	TOO MANY OPEN FILES ON UNIT

The total number of logical links allowed for the local node has been reached. The system manager can set a limit on the total number of links. See the system manager if this limit is inadequate. A retry may succeed.

18	ILLEGAL SYS( ) USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Issue a Remove Receiver call and reissue the Declare Receiver call.

Possible Errors (Cont.)

errval	Meaning
	o An invalid value was detected in one of the following fields: - Remote program field in the Connect Data Block. - One of the three access control fields in the Connect Data Block. - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255. - The local link modifier (*llmod*) value must be 0, 1, or 2. - The remote descriptor length (byte 10 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).
22	DISK PACK IS LOCKED OUT Session Control cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 172 bytes and 188 bytes for a Connect Initiate Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a Connect Initiate Message directed to a program named NETRCV at node MIAMI. The Connect Data Block uses a format 1 name to address the remote program. ECL supplies the local program identification and no access control fields are used, so the rest of the Connect Data Block is filled with spaces. A ULA of 27 is established and the receive maximum is set to 512. LLMOD is set to indicate that the local program requests message flow control. Message data is sent following the Connect Data Block. For example:

(Data Division)

```
01     NULLNUM PIC 9(4) COMP VALUE IS 0.
01     NULL REDEFINES NULLNUM PIC X.
01     CONNECT-BUF.
       02    CDB.
             03    REM-NODE PIC X(6).
             03    REM-PROG.
                   05    FMT PIC 9(4) COMP VALUE IS 1.
                   05    LEN PIC 9(4) COMP.
                   05    LEN-REV REDEFINES LEN.
                         07    HIGH-BYTE PIC X.
                         07    LOW-BYTE PIC X.
                   05    DESCRIP PIC X(16) VALUE IS "NETRCV".
             03    LOC-PROG PIC X(20).
             03    ACCESS-FIELDS PIC X(126).
             03    MESSAGE PIC X(7) VALUE IS "OPTION2".
01     RIB PIC 9(4) COMP VALUE IS 0.
01     TEMP PIC X.
01     STAT PIC 9(4) COMP.
01     ULA PIC 9(4) COMP VALUE IS 27.
01     RMAX PIC 9(4) COMP VALUE IS 512.
01     LLMOD PIC 9(4) COMP VALUE IS 2.
01     BUFLEN PIC 9(4) COMP VALUE IS 127.
01     BUFOFF PIC 9(4) COMP VALUE IS 0.
       .
       .
       .
```

(Procedure Division)

```
       MOVE "MIAMI" TO REM-NODE.
       MOVE 6 TO LEN.
       MOVE HIGH-BYTE TO TEMP.
       MOVE LOW-BYTE TO HIGH-BYTE.
       MOVE TEMP TO LOW-BYTE.
       CALL "NTCI" USING STAT, ULA, LLMOD, RMAX,
            CONNECT-BUF, BUFLEN, BUFOFF, RIB.
```

Send Connect Confirm Message: NTCC (DECnet)

The NTCC call accepts a remote program's request for a logical link. The call establishes the User Link Address (ULA) that the calling program uses to refer to the link. The Local Link Address (LLA) assigned by the local ECL identifies the particular link being accepted. The flow control option and the receive maximum are specified for the link. Up to 16 bytes of message data can be passed to the remote program.

COBOL Call Format

```
CALL "NTCC" USING errval,ula,lla,llmod,rmax,buffer,
                  buflen,bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTCC call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is a one-word COMP data item within the range 1 to 255. Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The Local Link Address (*lla*) is a one-word COMP data item that identifies the link being accepted. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in bytes 5-6 of the received Connect Initiate Message (see "Format of Received Connect Initiate Message").

llmod

The one-word COMP data item *llmod* indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0 No flow control
1 Segment flow control
2 Message flow control

See Chapter 4 for a full discussion of flow control. Note that both programs on a logical link select their own flow control independent of each other. The *llmod* value is the selection made by the calling program. If *llmod* is nonzero, the program must issue Network Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space but is not designed to process large messages in small pieces, can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive: MRCV" for a description of how to process large messages in small pieces.

buffer

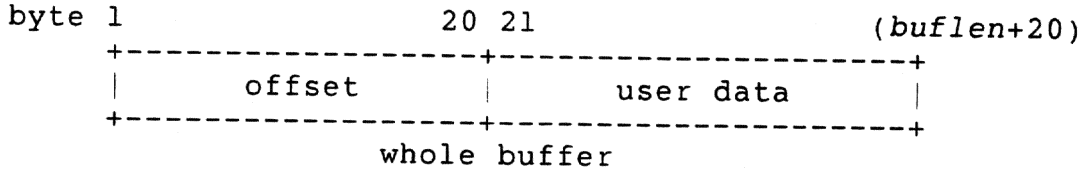
This argument is a data-name referring to the first character position (byte) of a buffer containing the optional user message data. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of this buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE

A logical link is currently active for the calling program with the same User Link Address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.

5	CAN'T FIND FILE OR ACCOUNT

The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.

10	PROTECTION VIOLATION

Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Confirm Message for the same logical link causes this error because the link is already established.

18	ILLEGAL SYS( ) USAGE

One of several possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- An invalid value was detected in one of the following fields:
  - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.
  - The local link modifier (*llmod*) value must be 0, 1, or 2.

31	ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Confirm Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, an NTCC call responds to a received Connect Initiate Message that was stored in data item RETBUF. The LLA is stored in bytes 5 and 6 of such an array and the data item LLA redefines these bytes to be referenced by that name. The call also establishes a ULA of 15, a receive maximum of 256 bytes, and indicates that the program requests message flow control. No user message data is sent so a dummy argument is given for the starting address, with zero values for the length and offset. For example:

```
(Data Division)

01 STAT PIC 9(4) COMP.
01 RETBUF.
   02 FILLER PIC X(4).
   02 LLA PIC 9(4) COMP.
   02 FILLER PIC X(34).
01 ULA PIC 9(4) COMP VALUE IS 15.
01 LLMOD PIC 9(4) COMP VALUE IS 2.
01 RMAX PIC 9(4) COMP VALUE IS 256.
01 Z-VAL PIC 9(4) COMP VALUE IS 0.
01 RIB PIC 9(4) COMP VALUE IS 0.
     .
     .
     .

(Procedure Division)

   CALL "NTCC" USING STAT, ULA, LLA, LLMOD,
    RETBUF, Z-VAL, Z-VAL, RIB
```

Send Connect Reject Message: NTCR (DECnet)

The NTCR call rejects a remote program's request for a logical link. The link is identified by the Local Link Address (LLA) established by the local ECL. You can send up to 16 bytes of user message data to the remote program as part of the Connect Reject Message.

COBOL Call Format

```
CALL "NTCR" USING errval,lla,reason,buffer,buflen, bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTCR call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

lla

The Local Link Address (*lla*) is a one-word COMP data item that identifies the link being rejected. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The *lla* is passed to the local program in bytes 5-6 of a received Connect Initiate Message (see "Format of Received Connect Initiate Message").

reason

DECnet/E NSP allows one of three reason code values for this one-word COMP data item: 0, 34, or 36.

Most user programs use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

buffer

This argument is a data-name referring to the first character position (byte) of a buffer containing the optional user message data. The data-name can refer to a single data item or an aggregate of data

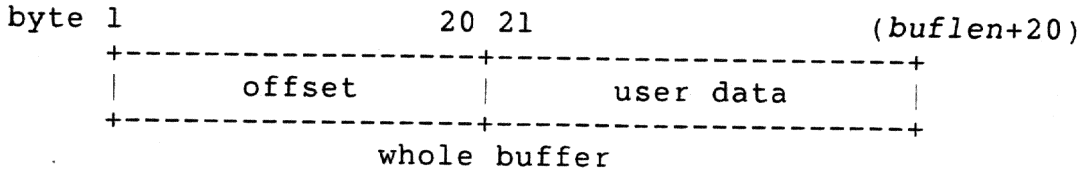
items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The message data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Reject Message for the same logical link returns this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *reason* code supplied with the Connect Reject Message was not 0, 34, or 36.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Reject Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In the following example, an NTCR call responds to a received Connect Initiate Message stored in the receiving buffer RETBUF. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data. For example:

(Data Division)

```
01 STAT PIC 9(4) COMP.
01 RETBUF.
   03 FILLER PIC X(4).
   03 LLA PIC 9(4) COMP.
   03 FILLER PIC X(34).
01 BUFFER PIC X(8) VALUE IS "TOO LATE".
01 BUFLEN PIC 9(4) COMP VALUE IS 8.
01 BUFOFF PIC 9(4) COMP VALUE IS 0.
01 REASON PIC 9(4) COMP VALUE IS 0.
01 RIB PIC 9(4) COMP VALUE IS 0.
     .
     .
     .
```

(Procedure Division)

```
   CALL "NTCR" USING STAT, LLA, REASON, BUFFER,
    BUFLEN, BUFOFF, RIB
```

Send Network Data Message: NTDM (DECnet)

The NTDM call transmits user data over an established logical link. The amount of user data you can transmit in one Network Data Message is called a segment. You can set a flag byte to indicate that the segment is also the end of a logical message. If the remote program requested segment or message flow control when the link was established, requests for data must be outstanding or this call results in an error.

COBOL Call Format

```
CALL "NTDM" USING errval,retbuf,ula,dmflgs,buffer,
                  buflen,bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTDM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument is a data-name referring to the first character position (byte) of a buffer to receive link status information returned by the call. You must specify this argument regardless of whether link status information is requested and it should refer to a buffer at least 40 bytes long.

ula

The *ula* is a one-word COMP data item giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The *dmflgs* argument is a one-word COMP data item that can be coded as the sum of three bit values:

dmflgs = rls + eom + bom

The low-order two bits of the *dmflgs* argument (*eom + bom*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, ECL uses these two bits to regulate the transmission of

logical messages as the remote program requests them. See Chapter 4 for a full discussion of flow control.

eom	*bom*	Meaning
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E ECL uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is passed to the remote system and program.

ECL also uses the *eom* flag to determine whether it permits a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* flag set, ECL assumes that the last message has not been completely sent and rejects a Disconnect Message. This is always done, regardless of the flow control option selected by the receiver.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, the *eom* flag should always be set.

The *rls* bit is used to request that link status information be returned to the *retbuf* buffer when the call has completed successfully:

rls = 128	Link status information is returned.
rls = 0	No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message (see "Format of Received Link Service Message"). If no link status information is requested, DECnet/E returns random data.

buffer

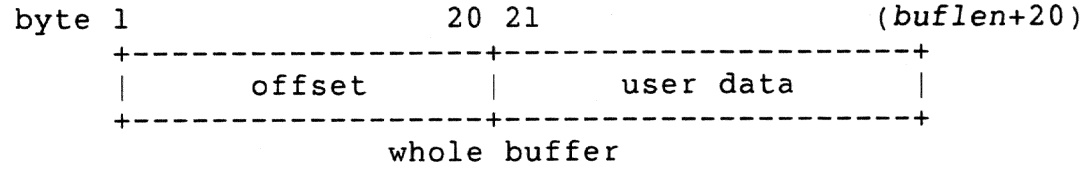
This argument is a data-name referring to the first character position (byte) of a buffer containing the user message data. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes to be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or by the remote ECL. It is passed on to the calling program as a transmit maximum (the *tmax* value in a received Connect Initiate or Connect Confirm Message). Thus, the value of *buflen* can range from 1 through *tmax*.

bufoff

This one-word COMP data item defines an offset, in bytes, from the start of the buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. One hundred bytes beginning at *buffer*+20 would be sent.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Data Message transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be in the range of 1 to 255.
19	DISK BLOCK IS INTERLOCKED Either the remote system has inhibited transmission on this link because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link). The calling program is notified with a Link Service Message when the condition clears (see Chapter 4).

Possible Errors (Cont.)

errval	Meaning
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example sends a Network Data Message over a logical link. The ULA is 20. The value for DMFLGS identifies the message data as the sole segment of a logical message and requests the return of link status information. The buffer for returned data is set up as a 40-byte data item named RETBUF.

The call sends 400 bytes of data from BUFFER. After the call, the program checks for an error. If none occurred, the program then determines if there are any pending messages. If so, the program branches to receive and process messages. If not, the program continues. For example:

(Data Division)

```
01      RETBUF.
        03   FILLER PIC X(2).
        03   CODE-ULA PIC 9(4) COMP.
        03   FILLER PIC X(4).
        03   REMLEN PIC 9(4) COMP.
        03   FILLER PIC X(2).
        03   MESLEN PIC 9(4) COMP.
        03   FILLER PIC X(6).
        03   L-AND-R-STAT PIC 9(4) COMP.
        03   L-AND-R-DRQ PIC 9(4) COMP.
        03   L-AND-R-IRQ PIC 9(4) COMP.
        03   DTM-AND-DTC PIC 9(4) COMP.
        03   LTM-AND-LTC PIC 9(4) COMP.
        03   MMAX-AND-MCNT PIC 9(4) COMP.
        03   MCNT REDEFINES MMAX-AND-MCNT.
             05  MMAXX PIC X.
             05  MCNTX PIC X.
        03   MULA PIC 9(4) COMP.
        03   FILLER PIC X(16).
01      MCNT-GOOD PIC 9(4) COMP.
01      WORK-WORD REDEFINES MCNT-GOOD.
        03   LOW PIC X.
        03   HIGH PIC X.
01      STAT PIC 9(4) COMP.
01      BUFFER PIC X(400).
01      BUFLEN PIC 9(4) COMP VALUE IS 400.
01      BUFOFF PIC 9(4) COMP VALUE IS 0.
01      ULA PIC 9(4) COMP VALUE IS 20.
01      EOM PIC 9(4) COMP VALUE IS 1.
01      BOM PIC 9(4) COMP VALUE IS 2.
01      RLS PIC 9(4) COMP VALUE IS 128.
01      DMFLGS PIC 9(4).
01      RIB PIC 9(4) COMP VALUE IS 0.
          .
          .
          .
```

(Procedure Division)

```
ADD EOM, BOM, RLS GIVING DMFLGS.
CALL "NTDM" USING STAT, RETBUF, ULA, DMFLGS,
     BUFFER, BUFLEN, BUFOFF, RIB.
IF STAT NOT EQUAL 0 GO TO ERR-PROC.
MOVE 0 TO MCNT-GOOD.
MOVE MCNTX TO LOW.
IF MCNT-GOOD NOT EQUAL 0 GO TO RECV-MSG.
```

Send Interrupt Message: NTIN (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message is placed at the head of the pending message queue, behind the first message (it can already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt messages are subject to flow control (see Chapter 4).

COBOL Call Format

```
CALL "NTIN" USING errval,retbuf,ula,inflgs,buffer,
                  buflen, bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTIN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument is a data-name referring to the first character position (byte) of a buffer to receive link status information returned by the call. You must specify this argument regardless of whether link status information is requested and it should refer to a buffer at least 40 bytes long.

ula

The *ula* is a one-word COMP data item specifying the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

The *inflgs* argument indicates whether link status information is to be returned to *retbuf* when the call has completed successfully. This field should be coded as follows:

```
inflgs = rls
```

where:

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message (see "Format of Received Link Service Message"). If no link status information is requested, random data is returned.

buffer

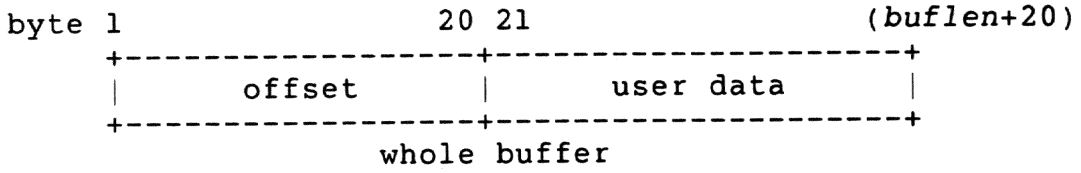
This argument is a data-name referring to the first character position (byte) of a buffer containing optional user message data. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes of user data to be sent from the buffer. Possible values of *buflen* range from 0 to 16.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a bufoff of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.

Possible Errors (Cont.)

errval	Meaning
19	DISK BLOCK IS INTERLOCKED The remote ECL has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message is delivered to the calling program when the condition clears. See Chapter 4 for a full discussion of flow control.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for an Interrupt Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTIN call for logical link 15 on RIB #9. The call sends the message data TIMEOUT and requests no link status information. However, a buffer must be defined and referenced, so an area named DUMMY is defined and used in the call. For example:

```
(Data Division)

01      ULA PIC 9(4) COMP VALUE IS 15.
01      DUMMY PIC X(40).
01      BUFFER PIC X(7) VALUE IS "TIMEOUT".
01      BUFLEN PIC 9(4) COMP VALUE IS 7.
01      BUFOFF PIC 9(4) COMP VALUE IS 0.
01      STAT PIC 9(4) COMP VALUE IS 0.
01      INFLGS PIC 9(4) COMP VALUE IS 0.
01      RIB PIC 9(4) COMP VALUE IS 9.
            .
            .
            .

(Procedure Division)

        CALL "NTIN" USING STAT, DUMMY, ULA, INFLGS,
             BUFFER, BUFLEN, BUFOFF, RIB.
```

Send Link Service Message: NTLS (DECnet)

The NTLS call:

- Requests data from a remote program over a flow controlled logical link (see Chapter 4)
- Reenables incoming Interrupt Messages over a logical link (see Chapter 4)
- Obtains status information for the link

No user data is allowed with a Link Service Message.

COBOL Call Format

```
CALL "NTLS" USING errval,retbuf,ula,lsflgs,drcnt, ircnt,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTLS call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument is a data-name referring to the first character position (byte) of a buffer to receive link status information returned by the call. You must specify the argument in the call regardless of whether link status information is requested and it should refer to a buffer at least 40 bytes long.

ula

The *ula* is a one-word COMP data item giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The link service flags argument is a one-word COMP data item coded as the sum of two values: *lsflgs* = *rls* + *opr*. The low-order two bits (*opr*) indicate the purpose of the call:

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested message or segment flow control in its Connect Initiate or Connect Confirm Message for this logical link. If no flow control was requested, an NTLS call with *opr* = 0 fails with an error (*errval* = 18).

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* argument is interpreted as the number of Interrupt Messages requested (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

You can use the *rls* bit when *opr* = 0 or 1 to request that link status information be returned to the *retbuf* buffer when the call completes successfully:

rls = 128 Link status information is returned.
= 0 No link status information is returned.

Link status information is always returned when *opr* = 2. If link status information is requested, it is returned in the same format as a received Link Service Message (see "Format of Received Link Service Message"). If no link status information is requested, DECnet/E returns random data.

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for this end of the logical link, the value is interpreted as either a segment counter (segment flow control) or a logical message counter (message flow control). In either case, it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local ECL (see Chapter 4). The program returns an error if the *drcnt* specified increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is a one-word COMP data item. It is meaningful only when *opr* = 1 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Hence, *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling interrupts on the logical link. Note that Interrupt Messages can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call fails with an error (*errval* = 19).

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. ECL is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed causes this error.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS( ) USAGE One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *opr* value is not 0, 1, or 2. o The *opr* value is 0, but the program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message. o The data request count (*drcnt*) increased the count of outstanding segments or logical messages requested to more than 127. o The *opr* value is 1, indicating a request for an Interrupt Message, but the *ircnt* argument does not equal 1. o The *opr* value is 1, indicating a request for an Interrupt Message, but the outstanding request counter for Interrupt Messages already equals 1. o The *ula* specified is zero. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED The *opr* value is 1, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

Possible Errors (Cont.)

errval	Meaning
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example of the NTLS call requests five segments from the remote program over logical link 23 on RIB #9. Assume that the calling program requested segment flow control in its NTCI or NTCC call for this link. For example:

(Data Division)

```
01      STAT PIC 9(4) COMP.
01      DUMMY PIC X(40).
01      OPR PIC 9(4) COMP VALUE IS 0.
01      RLS PIC 9(4) COMP VALUE IS 128.
01      LSFLGS PIC 9(4) COMP.
01      ULA PIC 9(4) COMP VALUE IS 23.
01      DRCNT PIC 9(4) COMP VALUE IS 5.
01      IRCNT PIC 9(4) COMP VALUE IS 0.
01      RIB PIC 9(4) COMP VALUE IS 9.
          .
          .
          .
```

(Procedure Division)

```
        CALL "NTLS" USING STAT, DUMMY, ULA,
         LSFLGS, DRCNT, IRCNT, RIB.
```

Send Disconnect Message: NTDI (DECnet)

The NTDI call terminates an established logical link. Messages in the pending message queue remains but no more messages can be sent over the link. The User Link Address (ULA) is freed and can be used for another logical link. If the last Network Data Message sent did not have the end-of-message (EOM) flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call fails with an error.

Successful completion of an NTDI call assures the sender that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not, however, guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see the section "Send/Receive Disconnect Message" in Chapter 3). Otherwise, the NTDI call provides no particular advantage over an NTLA call in terminating a logical link (see the section "Send Disconnect Message").

Up to 16 bytes of user data can be sent with the Disconnect Message.

COBOL Call Format

```
CALL "NTDI" USING errval,ula,buffer,buflen,bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTDI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is a one-word COMP data item giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

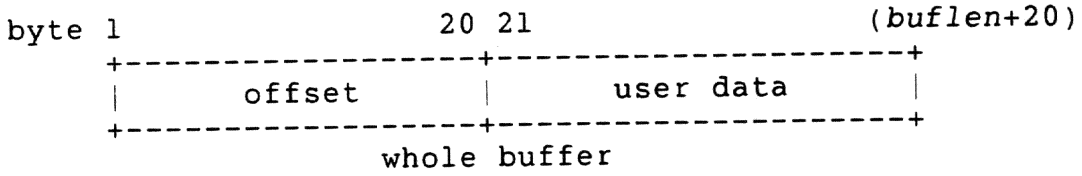
This argument is a data-name referring to the first character position (byte) of a buffer containing the optional user message data. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes of user data to be sent from the buffer. Possible values of *buflen* range from 0 to 16.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. If, for example, a program specifies a buffer 20 bytes long with an offset of 10 and a *buflen* of 16, 16 bytes are sent: the last 10 bytes of the buffer, and 6 bytes of whatever is beyond the buffer in memory.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS( ) USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be within the range 1 to 255.
19	DISK BLOCK IS INTERLOCKED The last segment sent by the local program over this logical link did not have the *eom* flag set. ECL does not permit a logical link to be disconnected unless all messages have been completely sent. Note that this test of the *eom* flag is done irregardless of the flow control option chosen by the local program.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example disconnects logical link 255 on RIB #9 and includes user data explaining the reason for the disconnect. Note that the remote program would have to be coded to recognize this message and take appropriate action. For example:

```
(Data Division)

01    ULA PIC 9(4) COMP VALUE IS 255.
01    STAT PIC 9(4) COMP.
01    BUFFER PIC X(6) VALUE IS "NOMORE".
01    BUFLEN PIC 9(4) COMP VALUE IS 6.
01    BUFOFF PIC 9(4) COMP VALUE IS 0.
01    RIB PIC 9(4) COMP VALUE IS 9.
        .
        .
        .

(Procedure Division)

      CALL "NTDI" USING STAT, ULA, BUFFER, BUFLEN, BUFOFF, RIB.
```

Send Link Abort Message: NTLA (DECnet)

The NTLA call terminates a logical link immediately. If there are still messages in the transmit queues waiting for acknowledgment from the remote ECL, the system discards them. These messages may or may not have been received by the remote system (see the section "Send/Receive Link Abort Message" in Chapter 3).

You can send up to 16 bytes of user data with the message. If you issue the NTLA call for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If you issue the NTLA call for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program.

COBOL Call Format

```
CALL "NTLA" USING errval,ula,buffer,buflen,bufoff,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the NTLA call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is a one-word COMP data item giving the User Link Address (in the range from 1 to 255) chosen by the calling program in the Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

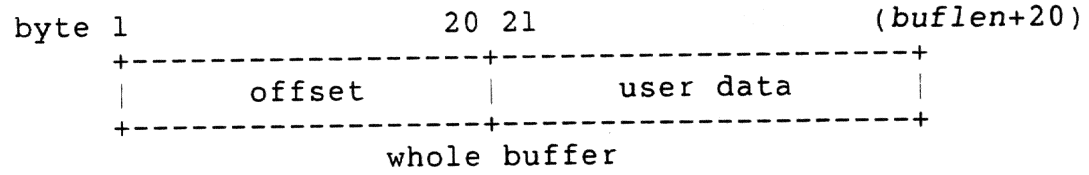
This argument is a data-name referring to the first character position (byte) of a buffer containing the optional user message data. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in the fields referenced by data-name. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the number of bytes of user data to be sent from the buffer. Possible values of *buflen* range from 0 to 16.

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be in the range of 1 to 255.

Possible Errors (Cont.)

errval	Meaning
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example aborts logical link 17 on RIB #9; no user data is sent. For example:

(Data Division)

```
01      STAT PIC 9(4) COMP.
01      ULA PIC 9(4) COMP VALUE IS 17.
01      DUMMY PIC X.
01      BUFLEN PIC 9(4) COMP VALUE IS 0.
01      BUFOFF PIC 9(4) COMP VALUE IS 0.
01      RIB PIC 9(4) COMP VALUE IS 9.
          .
          .
          .
```

(Procedure Division)

```
        CALL "NTLA" USING STAT, ULA, DUMMY, BUFLEN, BUFOFF, RIB.
```

Receive: MRCV (Local and DECnet)

The MRCV call retrieves a message from the calling program's pending message queue. It returns control parameters and user data, if any, to buffers defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer that receives the user data. The message data can be retrieved all at once or in portions, and portions can be discarded. The formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message" show what control information is returned for the various types of messages.

A Receive call can be selected to get:

- The first message in the queue
- The first message in the queue from either a network or a local sender
- The first message in the queue for a particular logical link

You can set a sleep flag and associated timer to indicate that the calling program is to wait if there are no appropriate messages pending in the queue.

COBOL Call Format

```
CALL "MRCV" USING errval,retbuf,rmod,sndr,qual,buffer,
                  buflen, bufoff, slptime,rib
```

Argument Descriptions

errval

The *errval* argument is a one-word COMP data item. It is set to zero if the MRCV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument is a data-name referring to the first character position (byte) of a buffer to contain the control information identifying the type and characteristics of the message received. This argument must refer to a buffer at least 40 bytes long. The format of the data returned depends on the message type, as described in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

rmod

The receive modifier (*rmod*) argument is a one-word COMP data item consisting of four bit values: *s* + *t* + *l* + *n*, described as follows:

s The sleep bit (*s*) defines what is to be done if there is no appropriate message in the queue for the Receive call to deliver -- that is, a message of the type requested with the *l*, *n*, *sndr*, and *qual* arguments. If *s* = 1, the program sleeps, suspending execution, until something happens. The *slptime* argument regulates the length of time that the program sleeps.

If the *s* bit is cleared to 0, the program does not sleep. If there are no appropriate messages in the queue the call terminates immediately with an error (*errval* = 5).

t The truncate bit (*t*) is set to 2 to discard any excess information that does not fit in the buffer specified in this call. Information could be left over if the value of *buflen* is smaller than the amount of user data in the message.

A 0 value for *t* indicates that such excess information is to be kept for retrieval on later Receive calls.

l The local sender bit (*l*) is set to 4 to select a message from a local sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *l* = 0, local selection is not requested. If both *l* and *n* are zero, the first message in the queue is returned.

n The network sender bit (*n*) is set to 8 to select a message from a network sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *n* = 0, network selection is not requested. If both *l* and *n* = 0, the first message pending in the queue is returned. If *l* = 4, and *n* = 8, local selection prevails.

sndr

The sender selection argument (*sndr*) is a one-word COMP data item. It selects a particular local sender or a particular logical link for this Receive call.

If *l* = 4 (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If *l* = 0, and *n* = 8 (network selection), a nonzero value for *sndr* is interpreted as a user link address. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either *l* = 4 or *n* = 8, the *qual* parameter has special meaning. The *sndr* and *qual* arguments are ignored if both *l* and *n* are zero.

qual

The sender selection qualifier (*qual*) argument is a one-word COMP data item. It is normally zero for user applications.

If *l* = 4 (local selection) and *sndr* = 0, any nonzero value for *qual* is a special Receive call, requesting a message from the system (job number 0). The RSTS/E utility ERRCPY uses this special case to process messages from the monitor error logging routines.

The *qual* argument is ignored if both *l* and *n* = 0, or if *sndr* $\neq$ 0.

Table 9-4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 9-4: Sender Selection Summary

rmod				
n	*l*	*sndr*	*qual*	Result
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	4	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times two. Only messages from that job are delivered on this MRCV call.
8	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a ULA. Only network messages from the designated logical link are delivered on this MRCV call.

buffer

This argument is a data-name referring to the first character position (byte) of the buffer to which user message data is to be delivered. The data-name can refer to a single data item or an aggregate of data items (for example, a level 01 group name). You can use any and all PICTURE categories in fields referenced by data-name. The format should, of course, reflect the type of data sent from the remote program. The data can be offset from the beginning of the buffer (see *bufoff*).

buflen

This one-word COMP data item defines the maximum number of bytes to be delivered to the buffer on this Receive call.

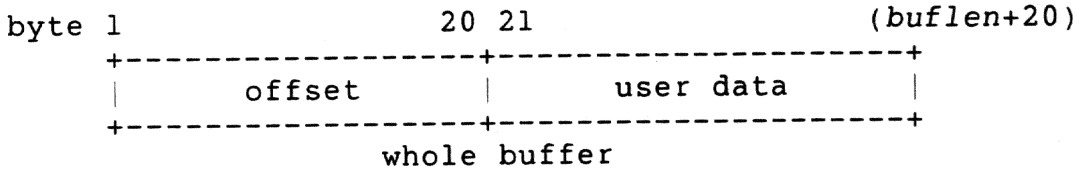
If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (*t* bit of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, then only the amount indicated by *buflen* is returned and the rest is saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining bytes 9-10 in the data returned to the *retbuf* buffer by the Receive call.

The number of bytes actually returned to the buffer is returned in bytes 13-14 in the *retbuf* buffer. See the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

bufoff

This one-word COMP data item defines an offset, in bytes, from the beginning of the buffer. For example, if *bufoff* = 20, the user message data is delivered beginning in byte 21 of the buffer.



Note

The extended COBOL interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. Up to 100 bytes beginning at *buffer*+20 could be delivered on the Receive call.

slptime

The *slptime* argument is a one-word COMP data item. It is significant only when the *s* bit of *rmod* = 1. In that case, a positive value for *slptime* defines the length of time (in seconds), that a program is to sleep when no appropriate message is in the queue. The program sleeps until:

- The sleep timer expires.
- Any new message is placed in the queue (not just an appropriate one).
- A delimiter is typed on a terminal opened by, or assigned to, the job.
- Log-ins are disabled. (This could occur if the system is being shut down).
- A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
- The job itself has opened the XM: (DMC11 driver) and a message is received for the job.
- The system time is changed.

In all cases, the program is awakened with an error (*errval* = 5) but is not passed a message. To obtain a pending message, the program must execute another Receive call. Because the program can be awakened by any of the conditions listed previously, you can check for pending messages by executing a MRCV call without a sleep.

rib

The receiver ID block (RIB) number. It must be a one-word COMP data item ranging from 0 to 255; values 128 through 255 are reserved for DIGITAL use.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's RIB quota.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT For a Receive call without sleep (*s* flag in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (*s* flag in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV call to retrieve any pending messages.
18	ILLEGAL SYS() USAGE No RIB exists for the calling program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example

In the following example, the first Receive call (on RIB #9) is issued for network messages with the sleep bit set and the timer set to five seconds. The truncate bit is 0 and the length argument is 0, so all the information that is available is saved for the second Receive call. The test for ERRVAL = 5 after the first Receive call, with sleep, ensures that a message is waiting when the second Receive call is issued. The BYTSGN subprogram is the sample program shown in the section "Coding" to convert signed one-byte binary data items to signed one-word COMP data items. The GO TO then checks the value identifying the type of message and branches accordingly. For example:

```
(Data Division)

01  RIB PIC 9(4) COMP VALUE IS 9.
01  S PIC 9(4) COMP VALUE IS 1.
01  N PIC 9(4) COMP VALUE IS 8.
01  RMOD PIC 9(4) COMP.
01  STAT PIC 9(4) COMP.
01  SLPTIME PIC 9(4) COMP VALUE IS 5.
01  Z-VAL PIC 9(4) COMP VALUE IS 0.
01  RETBUF.
    03 RETBUF-CHAR PIC X OCCURS 40 TIMES.
01  BUFFER PIC X(256).
01  BUFLEN PIC 9(4) COMP VALUE IS 256.
01  ABC PIC X.
01  DEF PIC 9(4) COMP.
          .
          .
          .
```

```
(Procedure Division)

    ADD S, N GIVING RMOD.
RCV. CALL "MRCV" USING STAT, RETBUF, RMOD, Z-VAL, Z-VAL,
     BUFFER, Z-VAL, Z-VAL, SLPTIME, RIB.
     IF STAT = 5 GO TO RCV.
     MOVE N TO RMOD.
     CALL "MRCV" USING STAT, RETBUF, RMOD, Z-VAL, Z-VAL,
      BUFFER, BUFLEN, Z-VAL, Z-VAL, RIB.
     IF STAT NOT EQUAL 0 GO TO ERR-RTN.
     MOVE RETBUF-CHAR(3) TO ABC.
     CALL "BYTE-VAL" USING ABC, DEF.
     ADD DEF, 10 GIVING JMP-VAL.
     GO TO LAPROC, DIPROC, LSPROC, INPROC, DMPROC,
      CRPROC, CCPROC, CIPROC, LMPROC
      DEPENDING ON JMP-VAL.
LAPROC. (processing for Link Abort Message)
DIPROC. (processing for Disconnect Message)
       .
       .
       .
```

Format of Received Local Data Message

The following format describes the information returned to the *retbuf* buffer for a Local Data Message. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive call (MRCV).

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -1, the function code for a Local Data Message.
4	1-byte binary	The job number (of the local sender) times two.
5	1-byte binary	The programmer number of the local sender (from the PPN).
6	1-byte binary	The project number of the local sender (from the PPN).
7	1-byte binary	Keyboard number of the sender or -1 if the sender is detached.
8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-40	whatever sent	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Format of Received Local Data Message with Privilege Mask

The following format describes the information returned to the *retbuf* buffer for a Local Data Message with Privilege Mask. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive call (MRPR).

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -11, the function code for a Local Data Message with Privilege Mask.
4	1-byte binary	The job number (of the local sender) times two.
5	1-byte binary	The programmer number of the local sender (from the PPN).
6	1-byte binary	The project number of the local sender (from the PPN).
7	1-byte binary	Keyboard number of the sender or -1 if the sender is detached.
8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument in the MRPR call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-28	Four 1-word COMP fields	The sender's privilege mask.
29-40	whatever sent	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 12 bytes.

Format of Received Connect Initiate Message

The following format describes the information returned to the *retbuf* buffer for a received Connect Initiate Message. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the data buffer specified in the Receive call. Figure 9-2 shows the format of a received Connect Data Block and Table 9-5 describes it in detail.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -2, the function code for a Connect Initiate Message.
4	--	Reserved.
5-6	1-word COMP	Local Link Address. Identifies the link for the local program's subsequent Connect Confirm or Connect Reject Message.
7-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23	1-byte binary	Remote link modifier. Indicates the flow control in effect for the remote program: If = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.

Format of Received Connect Initiate Message (Cont.)

Bytes	Data Type	Description
25-26	1-word COMP	Receive maximum (in bytes). The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This value is supplied by the local ECL and is determined by the size of the receive buffers that it allocates. The maximum size of the receive buffers is set for ECL by Network Management. The local program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a logical link (see the section "Send Connect Confirm Message").
27-28	1-word COMP	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

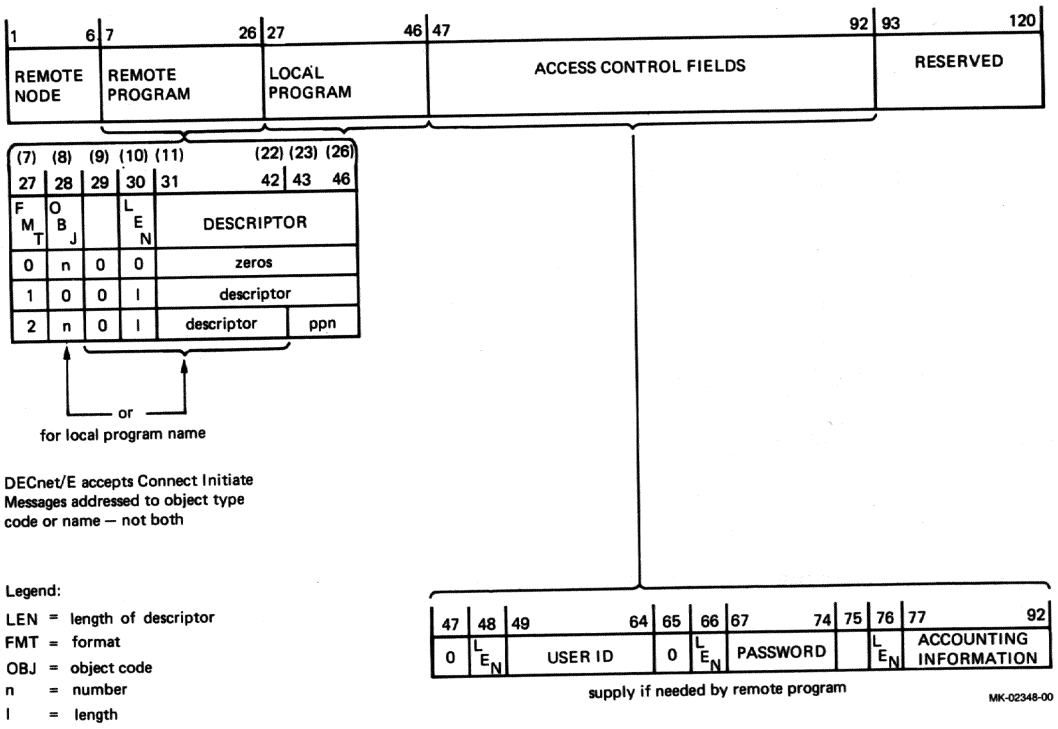


Figure 9-2: Format of Received Connect Data Block

Table 9-5: Format of Received Connect Data Block

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node from which the Connect Initiate Message was received. Names shorter than 6 characters are left-justified with space fill to 6 characters.
7-26	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of the following three formats.

Table 9-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 0 The remote program is identified by object type code alone. The fields are: **Bytes** — **Content** 7 — Zero, to indicate a format 0 name. 8 — Object type code of the remote program. 9-26 — Zero (not used).
	Format 1 The remote program is identified by a descriptor. The subfields are: **Bytes** — **Content** 7 — One, to indicate a format 1. 8 — Normally zero. This byte is usually zero unless the target DECnet system identifies the remote program with both name and object type code on outgoing connect requests. 9 — Reserved. 10 — Length (in bytes) of the descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26). 11-26 — Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Table 9-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 2 This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC. DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved.
10	Length (in bytes) of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor is left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E PPN.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E PPN.

Table 9-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
27-46	**Local Program Name.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Equal to 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.

Table 9-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
47-92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most precise checking done in a normal login on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system passes this information on to the receiving program.

Note

Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49-64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN at login.
65	Reserved.
66	Length of password information specified in the next field, 0 to 8 bytes.
67-74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login.

Table 9-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
75	Reserved.
76	Length of the accounting information specified in the following field, 0 to 16 bytes.
77-92	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.
93-120	Reserved.

Format of Received Connect Confirm Message

A Received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the receiving program. The link is identified by the User Link Address (ULA) specified by the receiving program in the Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. You can also return up to 16 bytes of user data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -3, the function code for a Connect Confirm Message.
4	1-byte binary	User Link Address.
5-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23	1-byte binary	Remote link modifier. Indicates the flow control in effect for the remote program: If = 0 No flow control = 1 Segment flow control = 2 Message flow control
24	--	Reserved.
25-26	1-word COMP	Receive maximum (in bytes). The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This is either equal to the *rmax* value specified in the Connect Initiate Message for this link, or a lesser value supplied by the local ECL, determined from the size of the receive buffers it allocates.

Format of Received Connect Confirm Message (Cont.)

Bytes	Data Type	Description
27-28	1-word COMP	Transmit maximum (in bytes). The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Format of Received Connect Reject Message

A Received Connect Reject Message is a rejection of the receiving program's request for a logical link. The link is identified by the User Link Address (ULA) previously established in the receiving program's Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. Rejections from ECL contain a reason code explaining the reason for the rejection and does not contain user data. Rejections from the remote program can contain up to 16 bytes of user data, delivered to the data buffer defined in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -4, the function code for a Connect Reject Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate Message.
5-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the t bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23-24	1-word COMP	Identifies the reason for the rejection (see Appendix B).
25-40	--	Reserved.

Format of Received Network Data Message

A Received Network Data Message contains user data from a remote program. The following format describes the information returned to the *retbuf* buffer. The user data accompanying the Network Data Message is delivered to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -5, the function code for a Network Data Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21	1-byte binary	Data message flags. Low-order bits indicate whether this is the beginning, middle, end, or sole segment of a logical message. If = 0 Middle segment of message = 1 First segment of message = 2 Last segment of message = 3 Sole segment of message DECnet/E uses bit 1 (set if this byte = 2 or 3) to determine the end of a logical message if the local program requested message flow control when the logical link was established. It does not use bit 0 but passes it on to the receiving program.
22-40	--	Reserved.

Format of Received Interrupt Message

The local ECL places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages. The following format describes the data returned to the *retbuf* buffer. You can deliver up to 16 bytes of user data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -6, the function code for an Interrupt Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-40	--	Reserved.

Format of Received Link Service Message

A Received Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call when the pending message queue is empty after a condition has cleared that previously inhibited the receiving program from sending (see the section "Flow Control" in Chapter 4). Status information with the same format as a Link Service Message can also be returned to the *retbuf* buffer on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The following format describes the information returned. No data is delivered to the data buffer with a Link Service Message.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -7, the function code for a Link Service Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	1-word COMP	Always zero for Link Service. (These bytes are the remainder field in other returned messages.)
11-12	--	Reserved.
13-14	1-word COMP	Always zero for Link Service. (These bytes are the length field in other returned messages.)
15-20	--	Reserved.

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
21	1-byte binary	Local status flags. See the section "Programming Background" for tips on testing bits. If bit 0 = 1, outgoing Network Data Messages are inhibited. A Link Service Message is delivered when the condition clears. If bit 1 = 1, outgoing Interrupt and Link Service Messages are inhibited. A Link Service Message is delivered when the condition clears. Bits 6 and 7 indicate the local ECL's backpressure status:

Bit 7	Bit 6	Meaning
0	0	Incoming Data Message flow is on.
0	1	Incoming Data Message flow is on but the local ECL turns this link off if another Data Message is received before the local program's receive queue is emptied.
1	0	Incoming Data Message flow has been turned off by the local ECL. The local program's receive queue must be emptied before the link is turned on.
1	1	Incoming Data Message flow has been turned off by the local ECL but flow is scheduled to turn on as soon as system buffers become available.

22	1-byte binary	Remote status flags. Only one bit is currently defined for this field. If bit 7 = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
23	1-byte binary	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	1-byte binary	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of interrupts requested but not yet received. This count never exceeds one.
25	1-byte binary	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	1-byte binary	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote end of the logical link, reflecting the actual count of interrupts requested but not yet sent.
27	1-byte binary	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.
28	1-byte binary	Data transmit queue count. The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote ECL.
29	1-byte binary	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
30	1-byte binary	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote ECL.
31	1-byte binary	Message maximum. The maximum number of incoming messages that are queued for the local program. This maximum is set by the program in its Declare Receiver call.
32	1-byte binary	Message count. The total number of messages currently in the pending message queue for the local program.
33	1-byte binary	Message count for this logical link. The number of messages currently in the pending message queue for the logical link identified by the *ula* in byte 4.
34-40	--	Reserved.

Format of Received Disconnect Message

A Received Disconnect Message indicates that an established logical link with a remote program has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The following format describes the data returned to the *retbuf* buffer. You can return up to 16 bytes of user message data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -8, the function code for a Disconnect Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-40	--	Reserved.

Format of Received Link Abort Message

A Received Link Abort Message indicates that a logical link has been terminated. The following format describes the data returned to the *retbuf* buffer. You can return up to 16 bytes of user message data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	1-byte binary	The value -9, the function code for a Link Abort Message.
4	1-byte binary	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-6	1-word COMP	Local Link Address, established by the local ECL for this link. It identifies the link if the remote program sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists under these conditions to identify the link being aborted.
7-8	--	Reserved.
9-10	1-word COMP	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* bit was set in the *rmod* argument in the MRCV call.
11-12	--	Reserved.
13-14	1-word COMP	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23-24	1-word COMP	Abort reason. If nonzero, the link was aborted by either the local or remote ECL and the reason is specified in these two bytes. The reasons that apply for a link abort are listed in Appendix B. If reason is zero, the link was aborted at the request of the remote program.
25-40	--	Reserved.

Chapter 10

Network Programming in FORTRAN

This chapter presents the specific calls, along with detailed formats, used to access the DECnet/E message services in the FORTRAN programming language.

Programming Background

To use the RSTS/E message services in FORTRAN, you code subroutine calls with the CALL statement. The calls reference external MACRO subroutines, stored in the system library, that must be linked to your compiled object module. Thus, the process involves coding, compiling, and linking. The following sections briefly describe these aspects of using DECnet/E in FORTRAN. Specific call formats are given in the sections "Declare Receiver: MDCL" through "Receive: MRCV."

Coding

The general format of the message calls in FORTRAN is:

```
CALL name (arg1,arg2, ...)
```

where:

name is the name of one of the 14 subroutines available for message services. Each subroutine processes a call at execution time by executing the following steps:

1. Checking for the proper number of arguments in the call.
2. Translating the call and arguments into the form recognizable to the RSTS/E monitor.
3. Passing control to the RSTS/E monitor.

RSTS/E processes the request and returns control to the user program. If an error occurs, RSTS/E returns an error value to an integer variable defined as an argument in each call.

arg... is the argument list for the particular call. The arguments are of two categories: input arguments and output arguments.

Input Arguments

Input arguments are those passed from your program to the DECnet/E MACRO subroutine. If the discussion of the argument calls for an integer value, you can specify any valid INTEGER*2 or INTEGER*4 variable, array element, expression, or constant for the argument. However, since no DECnet integer argument ever exceeds 32767, there is no reason for you to use INTEGER*4 values; they are unnecessary and require an extra word of storage.

If the discussion of the argument calls for a string, you can specify an alphanumeric literal ('LITERAL'), a Hollerith constant (7HCONSTANT), or a byte array containing the desired ASCII characters.

If the discussion of the argument calls for the starting address of a buffer, you can specify any valid FORTRAN variable, array, or array element. The contents of the buffer can, in some cases, require special formatting on your part. For example, the Send Connect Initiate Message call requires that you set up a Connect Data Block of 172 bytes. Some of the fields in the Connect Data Block can contain byte values ranging from 0 to 255. Note that you can set LOGICAL*1 values within this range with the normal replacement statement. For example, the following sequence works. However, if CDB(10) were used in some arithmetic calculation it would be treated as -1. For example:

```
LOGICAL*1 CDB(172)
        .
        .
        .
CDB(10) = 255
```

Output Arguments

Output arguments are those to which values are returned by the DECnet/E MACRO subroutine. There are only two possibilities:

- An error code returned as an integer; the argument must be an INTEGER*2 variable or array element.
- The starting address of a buffer containing returned information; the argument can be any valid FORTRAN array, variable, or array element.

In some cases, the buffer contains status information. Tips for accessing this information follow.

Tips for Returned Status Information

The Receive call and three send calls (send Network Data, Interrupt, and Link Service) return status and control information to a 40-byte user buffer. Some of this information consists of numeric byte values that, in DECnet terms, have values from 0 through 255. This differs from the FORTRAN treatment of numeric bytes as signed values ranging from -128 through +127. The easiest way to use these values in arithmetic conversions and comparisons is to convert them to positive integer values. A sample technique follows:

```
      LOGICAL*1          A(40)
      LOGICAL*1          A2(2,40)
      INTEGER            IA2(40)
      EQUIVALENCE        (A2(1,1),IA2(1))
      DO 20 I=1,40
20    A2(2,I) = 0
          .
          .
          .
(call that returns status information to buffer A)
          .
          .
          .
      DO 4000 I=1,40
4000  A2(1,I) = A(I)
```

In this example, the LOGICAL*1 array A is the buffer to which the 40-byte status information is returned. A2 is a two-dimensional byte array mapped to the one-dimensional integer array IA2 with an EQUIVALENCE statement. Since two-dimensional arrays in FORTRAN are stored by column, the first DO loop sets the high byte of each word of integer array IA2 to zero. The second DO loop sets the low byte of each word to the byte values in array A. The elements of array IA2 then are positive integer values corresponding to the elements of array A.

Note that not all the status information returned needs to be used in this way. Some values are signed bytes that can be directly referenced from array A in the previous example. Some are returned as integer values to which integer variable names can be directly assigned with an EQUIVALENCE statement.

Compiling

You can compile FORTRAN source code that includes DECnet subroutine calls with either one of two FORTRAN compilers available on RSTS/E systems:

- The FORTRAN compiler (for code to be run under the RT11 Run-Time System)
- The F77 (FORTRAN-77) compiler (for code to be run under the RSX Run-Time System)

Complete instructions on how to compile using the FORTRAN compiler are given in the *RT-11/RSTS/E FORTRAN IV User's Guide*.

Linking

As with the compilers, two linkers are available on RSTS/E systems. Use LINK with object modules produced by the FORTRAN compiler. LINK resolves references to the message subroutines from the system library ($SYSLIB.OBJ). See the *RSTS/E FORTRAN IV Utilities Manual* for a description of LINK.

To link object modules produced by F4P, use the RSTS/E Task Builder (TKB). The Task Builder automatically resolves references to the message subroutines from the system library (LB:SYSLIB.OLB). See the *RSTS/E Task Builder Reference Manual* for a description of the Task Builder.

Declare Receiver: MDCL (Local and DECnet)

You must execute the MDCL call before a program can receive any messages or send any network messages. Note that local messages can be sent without first executing this call. The call defines an identifying object type or name (or both) and any restrictions on incoming messages. DECnet/E associates this information with the RSTS/E job number, setting up a Receiver ID Block (RIB) for the job (see the section "DECnet/E Use of the Declare Receiver Call" in Chapter 3). It remains in effect until a MREM call (Remove Receiver) is executed for the job.

FORTRAN Call Format

```
CALL MDCL(errval,name,objtyp,access,0,mmax,lmax,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable or array element. This variable is set to zero if the MDCL call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

name

The logical name is a string consisting of one to six characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). The name is left-justified in the string and (if necessary) padded to six characters with spaces. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), the string should contain six spaces.

The name is used to identify the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code, if one is declared.

For local or network programs to access the calling program by logical name, the name must be nonnull and unique to the node. That is, only one program at the node can declare its identity with that name at any given time. The End Communication Layer (ECL) uses the logical name to identify the calling program for queuing messages from local or network programs. If the logical name is null, only network access by object type is allowed (see the discussion of *name*, *objtyp*, and *access*).

objtyp

This integer value specifies the object type code, another form of network addressing. As discussed in the section "Declaring Identity" in Chapter 3, the object type code defines some service the calling program performs. If the program issuing the MDCL call is addressed by object code alone (*name* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environment. Unlike the logical name, the object type code need not be unique within the node -- multiple copies of a program can declare their identity with the same type code simultaneously. If *name* is null, *objtyp* cannot be zero. See Chapter 4 for a detailed discussion on network addressing.

Acceptable values for the object type code range from 0 through 255:

- The value 0 indicates that object code addressing is not being used by the calling program
- The range from 1 to 127 is reserved for DECnet use (see Appendix A)
- The range from 128 to 255 is available for definition and use by a network installation.

access

This integer value is the sum of five bit values (*l* + *p* + *n* + *o* + *s*) that are used to determine access to the declaring program and to modify certain aspects of the message operations.

Three of these bit values (*l*, *p*, and *n*) limit the types of senders that can queue messages for the calling program. However, they do not limit the messages that the program can send.

If *l* = 0, messages from local senders are not queued for the calling program. If *l* = 1, messages from local senders are queued. Note that in this context, local senders who use the network functions are considered network senders.

If *p* = 0, incoming messages from local senders with and without privilege are queued for the calling program. If *p* = 2, incoming messages from local senders are queued only if the sender has SEND privilege. (This bit is ignored if *l* = 0, that is, if no local senders are allowed.)

If *n* = 0, Session Control rejects incoming requests for logical links. If *n* = 4, incoming Connect Initiate Messages are passed on to the receiver, subject to the declared link maximum (*lmax*). In this case, the program must provide its own protection from unauthorized access.

Table 10-1 summarizes the *l*, *p*, and *n* values for access.

Table 10-1: Types of Receiver Access

Access Code (l+p+n)	Network Logical Links	Local Senders with Privilege	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit value *o* is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If *o* = 0, incoming requests for logical links do not affect the program's link maximum. However, if *o* = 8, after queuing an incoming Connect Initiate Message to the receiver, Session Control sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 8, a program prevents the possibility of a Connect Initiate Message being queued to it after its one logical link is disconnected and before it issues a Remove Receiver call. For example, the DECnet/E FAL utility sets the one-shot bit to 8 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote system, Session Control queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that the first link is broken and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit value *s* is used to modify the function of the RSTS/E conditional Sleep monitor call. If *s* = 0, any unreceived messages queued for the program will block the execution of a conditional Sleep call. This is the normal RSTS/E function. However, if *s* = 16 the RSTS/E monitor does not check the program's message queue when determining whether it should suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can still block the sleep but a pending message does not.

name*, *objtyp*, and *access

Session Control checks the *name*, *objtyp*, and *access* arguments to make sure that they are consistent. For example, a null logical name and an access code allowing local senders are inconsistent, since local senders can address the program by logical name. The system checks these arguments as shown in Table 10-2. If the arguments are invalid, the system returns an error code of 18 in the *errval* argument. If the arguments are valid, the system allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (*errval* = 4). A retry may succeed.

Table 10-2: System Validation of Name, Object, and Access Parameters

Access Code (l+p+n)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local (only)	Ignored	Must be nonnull and unique.
4,6	Network (only)	Zero	Must be nonnull and unique.
		Nonzero	If *name* is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If *name* is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.
5,7	Network or Local	Any value	Must be nonnull and unique so local senders can refer to the calling program by logical name. Network senders can use the logical name or object type (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

mmax

The message maximum (*mmax*), an integer value ranging from 0 to 255, limits the number of messages that are queued for the calling program.

ECL keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the maximum set by a receiver:

- Incoming Local Data Messages are no longer queued for the receiving program. An error is returned to the local sender.
- Incoming Connect Initiate Messages cause ECL to determine whether another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
- An incoming Network Data Message causes ECL to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message. This forces the remote system to retransmit when flow is reenabled. See the section "Backpressure Flow Control" in Chapter 4 for further details.

lmax

The link maximum (*lmax*), an integer value ranging from 0 to 255, limits the number of incoming requests for logical links for the declaring program. If the number of links currently active is greater than or equal to *lmax* when a remote Connect Initiate Message is received for the calling program, the local ECL determines whether another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting *n* = 0 in the *access* argument. That is, Session Control rejects all incoming requests for logical links to the calling program. The *lmax* argument does not, however, limit the number of logical links initiated by the program. That is, it does not stop the program from sending Connect Initiate Messages. You must have SYSIO privilege to set a link maximum other than zero.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE

The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the last case, issue a Remove Receiver call followed by another Declare Receiver call. (It is common practice to code a Remove Receiver call immediately before a Declare Receiver call.)

4	NO ROOM FOR USER ON DEVICE

There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.

10	PROTECTION VIOLATION

The calling program must have SYSIO privilege at the time the Declare Receiver call is executed.

16	NAME OR ACCOUNT NOW EXISTS

The logical name specified is being used by another job.

18	ILLEGAL SYS() USAGE

The parameters specified are inconsistent. See the discussion of *logical name*, *object*, and *access*.

Example

The following example shows a Declare Receiver call on RIB #0 allowing both network and local access. Both a logical name (BUS) and an object type code (142) are specified. The buffer maximum is set to 512 and the message maximum is set to 5. No more than 512 bytes of user message data from incoming Local Data Messages are be stored in monitor pool space. If more than five messages accumulate for the program, an incoming request for a connection causes ECL to check to determine whether another copy of the program can be started automatically. If not, Session Control rejects the link. Incoming data messages at this point also causes ECL to inhibit incoming message flow on all the program's logical links because of the backpressure condition. The link maximum is set to 2. When the program has established two logical links, an incoming Connect Initiate Message causes ECL to check to determine whether another copy of the program can be started. If not, Session Control rejects the link. For example:

```
INTEGER ERRVAL,OBJTYP,ACCESS,BMAX,MMAX,LMAX,RIB
        .
        .
        .
OBJTYP = 142
ACCESS = 5
MMAX = 5
LMAX = 2
RIB = 0
CALL MDCL(ERRVAL,"BUS    ",OBJTYP,ACCESS,0,MMAX,LMAX,RIB)
IF ERRVAL .NE. 0 THEN 1200
```

Remove Receiver: MREM (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. Session Control discards all pending messages and aborts any active logical links. Execute this call at the completion of message processing to prevent unwanted messages from accumulating in the queue of pending messages.

Programs with JOBCTL privilege can remove other RSTS/E jobs with this call, although normally only system utility programs would do this.

FORTRAN Call Format

```
CALL MREM(errval,jobx2,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MREM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

jobx2

A zero value for *jobx2* removes the calling job. Programs with JOBCTL privilege can remove another local RSTS/E job by specifying job number times two for this integer argument.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
10	PROTECTION VIOLATION The caller does not have JOBCTL privilege and attempted to remove another job (that is, *jobx2* is nonzero).
18	ILLEGAL SYS() USAGE The *jobx2* argument was odd. *Jobx2* must be zero to remove the calling program or job number times two to remove another job.

Example

The following example shows a typical Remove Receiver call removing the calling job's RIB #26 from the system's list of declared receivers:

```
INTEGER ERRVAL,RIB
RIB = 26
        .
        .
        .
CALL MREM(ERRVAL,0,RIB)
```

Get Local Node Parameters: NTLN (DECnet)

The NTLN call returns information to the calling program concerning the local node's network parameters. Data returned includes the node name or node number, the node address, and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information. See the section "Automatic Job Start-Up" in Chapter 4 for a discussion of the default user's account.

FORTRAN Call Format

```
CALL NTLN(errval,buffer,buflen,bufoff)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

buffer

This argument defines the starting address of the buffer to receive the local node information. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *buffer*. When the NTLN call completes successfully, data is returned to the buffer in the following format:

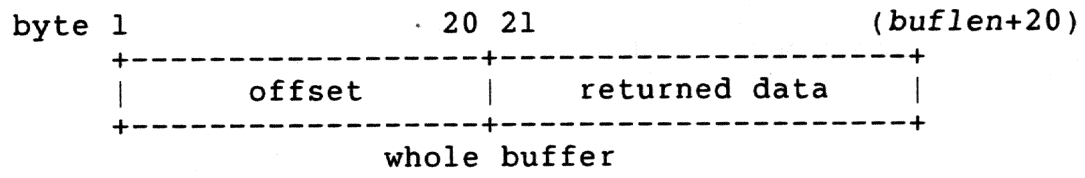
Bytes	Contents
0-5	Local node name, consisting of 1 to 6 uppercase, alphanumeric ASCII characters. The name contains at least one alphabetic character. A name shorter than 6 characters is left-justified and padded to 6 characters with spaces.
6-13	Reserved
14-15	Local default PPN. Byte 14 = programmer number; byte 15 = project number. If no default account is defined, both bytes are zero.

buflen

This integer argument defines the number of bytes to be returned to the buffer specified with *buffer*. If *buflen* is less than 16 bytes, the call returns only as much data as there is room for. For example, if *buflen* = 8 bytes, the call returns only the local node name and address. Sixteen bytes are required to receive all the local node parameters.

bufoff

This integer value defines an offset, in bytes, from the address specified in *buffer*. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if offset = 20, the returned data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of 10. If the actual buffer, beginning at *buffer* was only 16 bytes long, the returned data would overwrite 10 bytes beyond the end of the buffer.

Possible Errors

errval	Meaning
62	NO RUN-TIME SYSTEM

DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

66	MISSING SPECIAL FEATURE

DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTLN call used to obtain all the local node parameters. The node name is to be stored in the LOGICAL*1 variable NAME, and the default project-programmer number, in the LOGICAL*1 variables PROJ and PROG. For example:

```
LOGICAL*1 BUFFER(16),NAME(6),NUMBER(6),PROJ,PROG
INTEGER BUFLEN,BUFOFF,ADDRSS,ERRVAL
EQUIVALENCE (BUFFER(1),NAME(1))
EQUIVALENCE (BUFFER(15),PROG)
EQUIVALENCE (BUFFER(16),PROJ)
        .
        .
        .
BUFLEN = 16
BUFOFF = 0
CALL NTLN(ERRVAL,BUFFER,BUFLEN,BUFOFF)
```

Log User Event: NTEV (DECnet)

The NTEV call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

Note

This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the experienced network user and is not intended for normal network programs.

You need SYSIO and SEND privileges to use NTEV.

FORTRAN Call Format

```
CALL NTEV(errval,evmod,evcls,evtyp,entyp,enval,
          buffer,buflen,bufoff)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTEV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

evmod

This integer argument specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with a status return of NOROOM (*errval* = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a Missed event message, and the call terminates with a successful status (*errval* = 0).

Note that in the second case, the user data is not logged. The system only logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet internal events when the event queue is full or there are no more small buffers.

evcls

This integer argument specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480.

evtyp

This integer argument specifies the type of event within the class specified by *evcls*. The event type must be in the range of 0 to 31.

entyp

This integer argument specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

-1	No specific entity
0	Node
1	Line
3	Circuit

enval

This integer argument associates a value to the entity specified by the *entyp* argument. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

buffer

This argument defines the starting address of the buffer containing the optional user data to be processed with the event. If a nonzero offset is specified (see *bufoff*), the offset is added to the starting address of the buffer.

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value.

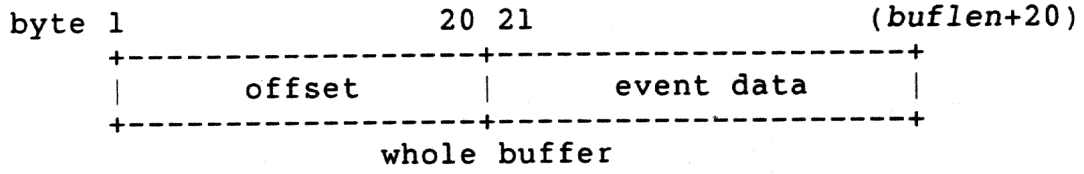
You can use DECnet parameter types if the event concerns a node, a circuit, or a line. In this case, the event logger outputs standard descriptive text for the specific parameter. If you do not use a DECnet parameter type, parameter types in the range of 3900 to 4095 can be specified. See the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for further information on event definitions and parameters. Up to 200 bytes of data can be passed to the event logger for processing.

buflen

This integer value defines the number of bytes of data in the buffer specified with *buffer*. Up to 200 bytes of data can be passed. If no data is specified, *buflen* must be 0.

bufoff

This integer value defines an offset, in bytes, from the address specified in *buffer*. The value of *bufoff* is added to *buffer* to define the beginning address of the returned data. For example, if *bufoff* = 20, the data to be processed begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *buffer*-100.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod*=0.)
5	CAN'T FIND FILE OR ACCOUNT The event logger is not running. Have the system manager start the event logger.
10	PROTECTION VIOLATION The calling program did not have SYSIO or SEND privileges at the time the NTEV call was issued.

Possible Errors (Cont.)

errval	Meaning
18	ILLEGAL SYS() USAGE The parameters passed are either inconsistent or invalid.
66	MISSING SPECIAL FEATURE DECnet/E was not installed.The network functions cannot be executed.

Example

The following example shows the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of an *11a* parameter number (2130) and value. A Missed event message is logged if the event itself cannot be. For example:

```
LOGICAL*1 DATBUF(30),DATLEN
INTEGER BUFOFF,BUFLEN,DATPRM,DATVAL
INTEGER EVMOD,EVTYP,EVCLS,ENTYP,ENVAL,ERRVAL
EQUIVALENCE (DATBUF(7),DATPRM),(DATBUF(9),DATLEN)
EQUIVALENCE (DATBUF(10),DATVAL)
        .
        .
        .
BUFOFF = 6
BUFLEN = 5
DATPRM = 2130
DATLEN = 2
DATVAL = 40960
EVMOD = 1
EVTYP = 480
EVCLS = 3
ENTYP = 0
ENVAL = 135
CALL NTEV(ERRVAL,EVMOD,EVCLS,EVTYP,ENTYP,ENVAL,DATBUF,BUFLEN,BUFOFF)
```

The NTEV call in the previous example generates the following event message:

```
Event type 480.3
Occurred 24-Sep-86 15:31:39.7 on node 3.135 (EDERE)
Node address 3.135 (EDERE)
Parameter  2130 = 40960
```

Send Calls

There are ten separate send calls -- two local and eight network. Each of these calls causes Session Control to format a message that is then either queued to a local program or transmitted across the network to a remote program.

Send Local Data Message: MSLD (Local)

The MSLD call sends up to 532 bytes of user data to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, the calling program must have SEND privilege or the MSLD fails with an error (*errval* = 10). The receiving program is identified in the MSLD call either by its job number or by the logical name with which it declared itself a receiver.

FORTRAN Call Format

```
CALL MSLD(errval,job,name,buffer,buflen,bufoff,
          parptr,parlen)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MSLD call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

If *job* = 0, the call uses the logical name to determine the receiver. If *job* = 128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job* = 8 directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name

- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The *name* argument identifies the intended local receiver when the *job* argument = 0. The intended receiver must have declared its identity with this name in its Declare Receiver call. The name consists of one to six characters, left-justified and, if necessary, padded with spaces. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

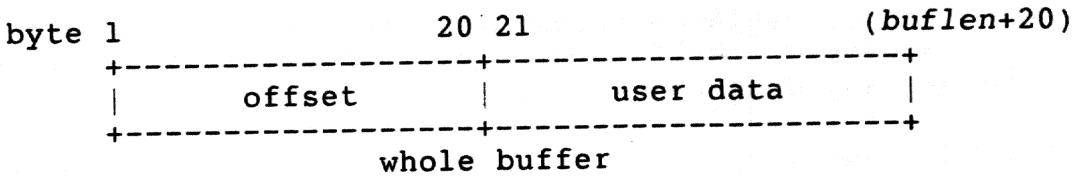
This argument defines the starting address of a buffer containing the message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by the *buffer* argument.

buflen

This integer argument defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the message data to be sent. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSLD call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument defines the starting address of a buffer containing up to 20 bytes of parameter message data to be sent to the local receiver. This data is delivered separate from the message data in the *buffer* area. For example, a RSTS/E FORTRAN program issuing a Receive call that returns a Local Data Message receives the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This integer value defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 20.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	CAN'T FIND FILE OR ACCOUNT The intended local receiver could not be located in the system's list of declared receivers. The receiving program must have declared itself a receiver (with a Declare Receiver call) before any data can be transmitted to it.
10	PROTECTION VIOLATION Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	ILLEGAL SYS( ) USAGE The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	ILLEGAL BYTE COUNT FOR I/O Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512; the *parlen* argument can range from 0 through 20.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Example

The following example shows a local send to a program that has issued a Declare Receiver call with the logical name LOCAL. A total of 156 bytes are sent from array A. No parameter message data is sent. For example:

```
BYTE A(156)
INTEGER BUFLEN,BUFOFF,ERRVAL
        .
        .
        .
BUFLEN = 156
BUFOFF = 0
CALL MSLD(ERRVAL,0,'LOCAL ',A,BUFLEN,BUFOFF,0,0)
```

Send Local Data Message with Privilege Mask: MSPR (Local)

The Send Local Data Message with Privilege Mask call sends a data message plus a privilege mask supplied by the monitor. This subfunction provides a method for a program to tell another program about a job's current privileges and guarantees that the data cannot be falsified.

A local job can send its privileges to a declared receiver in the same way as the Send Local Data Message call. See the discussion in the previous section.

FORTRAN Call Format

```
CALL MSPR(errval,job,name,buffer,buflen,bufoff,
          parptr,parlen)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MSPR call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

job

If *job* = 0, the call uses the logical name to determine the receiver. If *job* = 128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job* = 8 directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver

only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

name

The *name* argument identifies the intended local receiver when the *job* argument = 0. The intended receiver must have declared its identity with this name in its Declare Receiver call. The name consists of one to six characters, left-justified and, if necessary, padded with spaces. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore).

Leading or embedded spaces are invalid. Declared logical name of local program to receive the message: 1 to 6 characters. Left-justified with space fill to 6 characters. If *job* is nonzero and

you specify a receiver name, the send will succeed only if the name is associated with the job number.

buffer

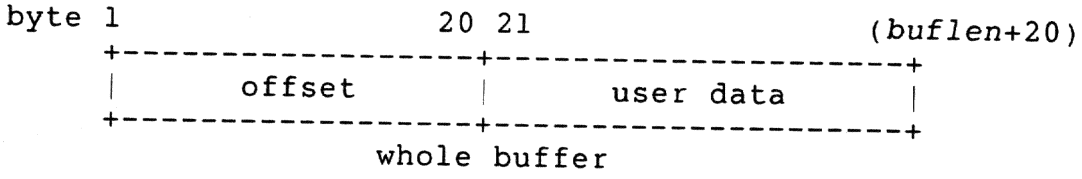
This argument defines the starting address of a buffer containing the message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by the *buffer* argument.

buflen

This integer argument defines the number of bytes to be sent from the buffer. Possible values for *buflen* range from 0 through 512.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the message data to be sent. For example, if offset = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSPR call would send 512 bytes beginning at *buffer*+1000.

parptr

This argument defines the starting address of a buffer containing up to 12 bytes of parameter message data to be sent to the local receiver. This data is delivered separate from the message data in the *buffer* area. For example, a RSTS/E FORTRAN program issuing a Receive call that returns a Local Data Message receives the parameter data and the message data in two separate buffers defined in the Receive call.

parlen

This integer value defines the number of bytes to be sent from the parameter area defined by *parptr*. The *parlen* value can range from 0 through 12.

Possible Errors

This call returns the same errors as the Send Local Data Message call. See the previous section.

Send Connect Initiate Message: NTCI (DECnet)

The NTCI call requests a logical link to another program. The call establishes the User Link Address the calling program uses to refer to the link. The other program is identified with a Connect Data Block. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 bytes of user data can be sent with the NTCI call.

FORTRAN Call Format

```
CALL NTCI(errval,ula,llmod,rmax,buffer,buflen,
          bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is an integer value in the range of 1 to 255. Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI call is accepted by the remote ECL and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

llmod

The integer value *llmod* indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

0	No flow control
1	Segment flow control
2	Message flow control

Chapter 4 describes flow control in detail. Note that each program on a logical link selects its own flow control. The *llmod* value is the selection made by the calling program. If *llmod* is nonzero, the calling program must issue Send Link Service Message (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value. Note that the calling program is informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive (MRCV)" for a description of how to process large messages in small pieces.

buffer

This argument defines the starting address of a buffer containing the Connect Data Block and optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

The Connect Data Block identifies the target program for the requested logical link. Figure 10-1 shows the format of the Connect Data Block. Table 10-3 describes the fields within the block.

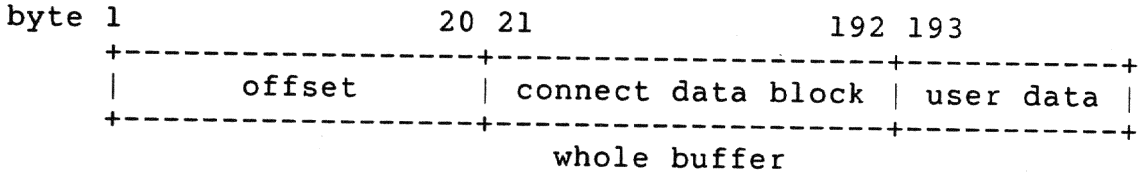
Up to 16 bytes of user data can follow the Connect Data Block.

buflen

This integer value defines the number of bytes to be sent. Since the Connect Data Block is 172 bytes long and the optional user data can be from 0 to 16 bytes long, acceptable values for *buflen* range from 172 through 188.

bufoff

This integer value defines an offset, in bytes, from the start of the buffer from which the data is to be sent. The offset value is added to the starting address of the buffer to define where the Connect Data Block begins. For example, if *bufoff* = 20, the Connect Data Block begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example specify a buffer 100 bytes long, an offset of 20 and a length of 188. In this case, 188 bytes would be sent; that is, the last 80 bytes of the buffer and 108 bytes of whatever is beyond the buffer in memory.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

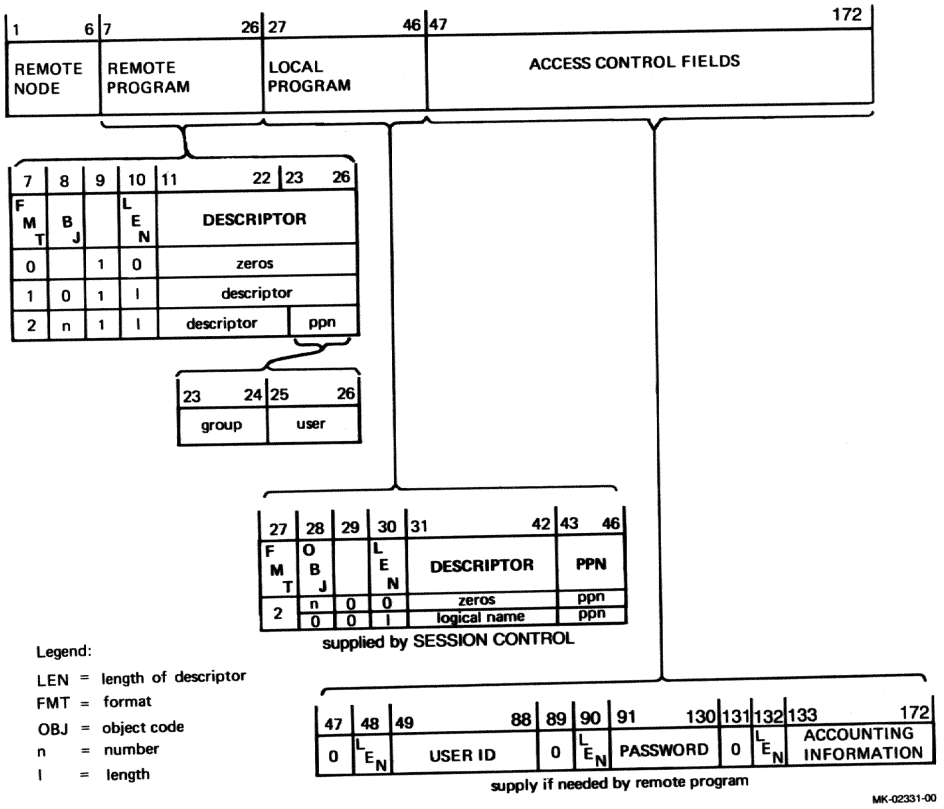


Figure 10-1: Format of Connect Data Block on Send Connect Initiate Message

Table 10-3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node to which the Connect Initiate Message is to be sent. Names shorter than 6 characters must be left-justified with space fill to 6 characters. Leading and embedded spaces are invalid. If no node name is specified (that is, the request is directed to a program at the local node), these 6 bytes contain an ASCII 0 and 5 spaces.
7-26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats: **Format 0** The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9	Must be one.
10-26	Zero (Not used).

Table 10-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	Format 1 The remote program is identified by a descriptor. The fields are: **Bytes** — **Content** 7 — One, to indicate a format 1 name. 8 — Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). 9 — Must be one. 10 — Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field. 11-26 — Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations. **Format 2** This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

Table 10-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The subfields are:
	Bytes / **Content**
	7 — Two, to indicate a format 2 name.
	8 — Object type code of the remote program, if applicable. If zero, the descriptor must be used.
	9 — Must be one.
	10 — Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
	11-22 — Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
	23-24 — Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E PPN.
	25-26 — Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E PPN.
27-46	**Local Program.** This field is filled in by the local session control using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

Table 10-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with an object code alone, the object code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The RSTS/E PPN under which the program is running is passed in bytes 43-46 as the group and user codes: the project number is in byte 43; the programmer number is in byte 45. Both are in binary format.
47-172	**Access Control Fields.** These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number or system password, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described.

Note

Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved -- must be zero.
48	Length of user identification information specified in the next field, 0 to 40 bytes.

Table 10-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
49-88	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, set the length (byte 48) to zero.
89	Reserved -- must be zero.
90	Length of password information specified in the next field, 0 to 40 bytes.
91-130	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, set the length (byte 90) to zero.
131	Reserved -- must be zero.
132	Length of the accounting information specified in the following field, 0 to 40 bytes.
133-172	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE The calling program already has a logical link with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NOT A VALID DEVICE The node named in the Connect Data Block is not known to the system.
14	DEVICE HUNG OR WRITE LOCKED The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	TOO MANY OPEN FILES ON UNIT The total number of logical links allowed for the local node has been reached. The system manager can set a limit on the total number of links. See the system manager if this limit is inadequate. A retry may succeed.
18	ILLEGAL SYS() USAGE One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the program to the remote system. Issue a Remove Receiver call and reissue the Declare Receiver call. o An invalid value was detected in one of the following fields: - Remote program field in the Connect Data Block. - One of the three access control fields in the Connect Data Block.

Possible Errors (Cont.)

errval	Meaning
	- The User Link Address (*ula*) given was zero. Valid values range from 1 to 255. - The local link modifier (*llmod*) value must be 0, 1, or 2. - The remote descriptor length (byte 10 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).
22	DISK PACK IS LOCKED OUT Session Control cannot create a new logical link because the physical line to the remote node or the local node itself has been scheduled for shutdown by the system manager.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 172 bytes and 188 bytes for a Connect Initiate Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a Connect Initiate Message directed to a program declared as a receiver with object type code 254 at node MIAMI. The Connect Data Block uses a format 2 name to address the remote program. Note that the replacement statement OBJTYP = 254 sets the correct byte in the Connect Data Block to 254, even though LOGICAL*1 values in FORTRAN are treated as signed values. ECL supplies the local program identification. No access control fields are used so the rest of the Connect Data Block is filled with zeros.

A user link address of 27 is established and the receive maximum is set to 512. LLMOD is set to indicate that the local program requests message flow control. The message data "OPTION2" is sent following the Connect Data Block. For example:

```
      LOGICAL*1 CDB(188),MSG(16),NODE(6),OBJTYP,FMT,LGT
      INTEGER BUFLEN,BUFOFF,ULA,LLMOD,RMAX,ERRVAL,RIB
      EQUIVALENCE (CDB(1),NODE(1)),(CDB(7),FMT),(CDB(8),OBJTYP))
      EQUIVALENCE ((CDB(173),MSG(1))
      DO 20 I = 1,188
   20 CDB(I) = 0
         .
         .
         .
      ULA = 27
      LLMOD = 2
      RMAX = 512
      FMT = 2
      OBJTYP = 254
      BUFLEN = 127
      BUFOFF = 0
      RIB = 16
      CALL NTCI(ERRVAL,ULA,LLMOD,RMAX,CDB,BUFLEN,BUFOFF,RIB)
         .
         .
         .
      DATA NODE/'M','I','A','M','I',' '/
      DATA MSG/'O','P','T','I','O','N','2'/
```

Send Connect Confirm Message: NTCC (DECnet)

The NTCC call accepts a remote program's request for a logical link. The call establishes the User Link Address (ULA) that the calling program uses to refer to the link. The Local Link Address (LLA) assigned by the local ECL identifies the particular link being accepted. The desired flow control option and the receive maximum for the link are also specified. Up to 16 bytes of message data can be passed to the remote program.

FORTRAN Call Format

```
CALL NTCC(errval,ula,lla,llmod,rmax,buffer, buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTCC call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The User Link Address (*ula*) is an integer value in the range of 1 to 255. Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The Local Link Address (*lla*) is an integer value that identifies the link being accepted. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in bytes 10-6 of the received Connect Initiate Message (see the section "Format of Received Connect Initiate Message").

llmod

The integer value *llmod* indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

0	No flow control
1	Segment flow control
2	Message flow control

See Chapter 4 for a detailed description of flow control. Note that both programs in a logical link select their own flow control independent of each other. The *llmod* value is the selection made by the calling program. If *llmod* is nonzero, the calling program must issue Send Link Service Message (NTLS) calls to request Network Data Messages from the other program.

rmax

A program that has limited buffer space and is not designed to process large messages in small pieces can impose a limit on the amount of user message data it is willing to receive on a logical link. The integer value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive (MRCV)" for a description of how to process large messages in small pieces.

buffer

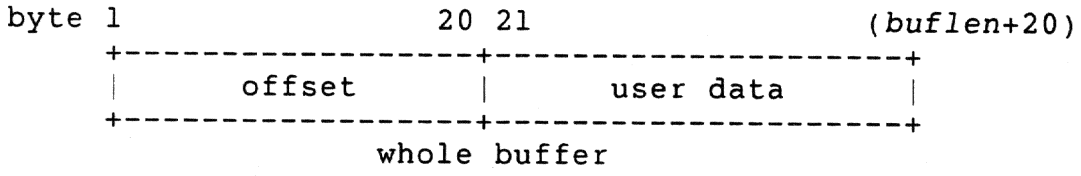
This argument defines the starting address of a buffer containing optional user message data. The data in the buffer is assumed to be in contiguous bytes. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the buffer argument. The offset value is added to the starting address of the buffer to define where the user data begins. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
3	ACCOUNT OR DEVICE IN USE A logical link is currently active for the calling program with the same User Link Address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Confirm Message for the same logical link causes this error because the link is already established.
18	ILLEGAL SYS() USAGE One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o An invalid value was detected in one of the following fields: - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255. - The local link modifier (*llmod*) value must be 0, 1, or 2.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Confirm Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

In the following example, an NTCC call responds to a received Connect Initiate Message that was stored in the byte array RETBUF. The LLA is stored in bytes 5 and 6 of this string and the EQUIVALENCE sets the integer variable LLA to access these bytes. The call also establishes a ULA of 15, a receive maximum of 256 bytes, and indicates that the program requests message flow control. No user message data is sent, so a dummy argument is given for the starting address with zero values for the length and offset. For example:

```
LOGICAL*1  RETBUF(40)
INTEGER ERRVAL,ULA,LLMOD,RMAX,LLA,RIB
EQUIVALENCE (RETBUF(5),LLA)
          .
          .
          .
ULA = 15
LLMOD = 2
RMAX = 256
RIB = 12
CALL NTCC(ERRVAL,ULA,LLA,LLMOD,RMAX,NOBUF,0,0,RIB)
```

Send Connect Reject Message: NTCR (DECnet)

The NTCR call rejects a remote program's request for a logical link. The link is identified by the Local Link Address (LLA) established by the local ECL. You can send up to 16 bytes of user message data to the remote program as part of the Connect Reject Message.

FORTRAN Call Format

```
CALL NTCR(errval,lla,reason,buffer,buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. If the NTCR call succeeds, this variable is set to zero. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

lla

The Local Link Address (*lla*) is an integer value that identifies the link being rejected. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in bytes 5-6 of a received Connect Initiate Message (see the section "Format of Received Connect Initiate Message").

reason

DECnet/E allows a program to specify one of the following three reason code values for this word: 0, 34, or 36.

Most user programs use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

buffer

This argument defines the starting address of a buffer containing the optional user message data. The data in the buffer is assumed to be in contiguous bytes. See the section "Coding" for a discussion of how

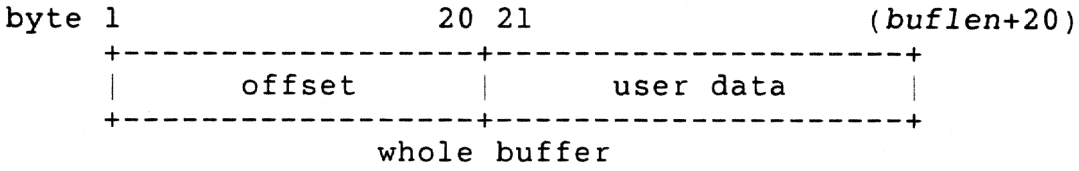
to set up buffers for message operations. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT

The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Reject Message for the same logical link returns this error.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *reason* code supplied with the Connect Reject Message was not 0, 34, or 36.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. It must be between 0 and 16 bytes for a Connect Reject Message.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A retry may succeed.

62 NO RUN-TIME SYSTEM

DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

66 MISSING SPECIAL FEATURE

DECnet/E was not installed. The network functions cannot be executed.

Example

In the following example, a NTCR call responds to a received Connect Initiate Message stored in the receiving byte array RETBUF. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data. For example:

```
LOGICAL*1 RETBUF(40)
INTEGER ERRVAL,LLA,REASON,BUFLEN,BUFOFF,RIB
EQUIVALENCE (RETBUF(5),LLA)
        .
        .
        .
BUFLEN = 8
BUFOFF = 0
REASON = 0
RIB = 24
CALL NTCR(ERRVAL,LLA,REASON,'TOO LATE',BUFLEN,BUFOFF,RIB)
```

Send Network Data Message: NTDM (DECnet)

The NTDM call transmits user data over an established logical link. The amount of data you can transmit in one Network Data Message is called a segment. You can set a flag to indicate whether the segment is the end of a logical message. If the remote program requested segment or message flow control when the link was established, requests for data must be outstanding or this call results in an error.

FORTRAN Call Format

```
CALL NTDM(errval,retbuf,ula,dmflgs,buffer,buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTDM call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the starting address of the buffer to receive link status information returned by the call. You need to specify this argument regardless of whether link status information is requested and it has to refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value specifying the User Link Address (in the range from 1 to 255) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The *dmflgs* argument is an integer value that can be coded as the sum of three bit values:

dmflgs = rls + eom + bom

The low-order two bits of the *dmflgs* argument (*eom + bom*) indicate whether the segment being sent in this Network Data Message is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, ECL uses these two bits to regulate the transmission of logical messages as the remote program requests them. See Chapter 4 for a full discussion of flow control.

eom	*bom*	Meaning
0	0	Middle segment of a logical message
0	1	First segment of a logical message
2	0	Last segment of a logical message
2	1	Sole segment of a logical message

DECnet/E NSP uses the *eom* bit to count logical messages transmitted to the remote program. The *bom* bit is not used in maintaining the logical message request count but is passed to the remote system and program.

NSP also uses the *eom* bit to determine whether it permits a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have the *eom* bit set, NSP assumes that the last message has not been completely sent and does not permit the disconnect. This is always done, without regard to the flow control option selected by the receiver.

Some DECnet implementations require that the *eom* flag be set to terminate an asynchronous I/O operation. It is suggested that when segment flow control or no flow control was requested by the remote program, the *eom* flag always be set.

The *rls* bit is used to request that link status information be returned to the *retbuf* buffer when this call has completed successfully.

rls = 128	Link status information is returned.
rls = 0	No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message. If no link status information is requested, DECnet/E returns random data.

buffer

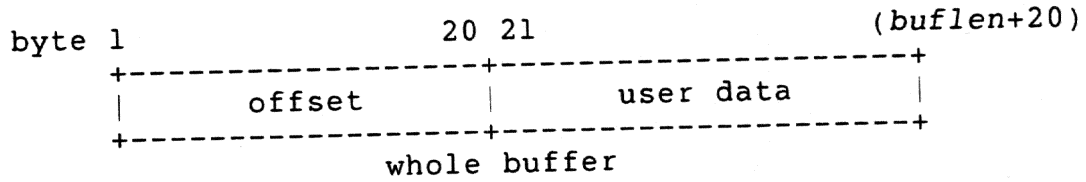
This argument defines the starting address of a buffer containing the user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer value defines the number of bytes tc be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or the remote ECL. It is passed on to the calling program as a transmit maximum (the *tmax* value in a received Connect Initiate or Connect Confirm Message. Thus, the value of *buflen* can range from 1 through *tmax*.

bufoff

This integer value defines an offset, in bytes, from the starting address of the buffer defined by the buffer argument. The value of *bufoff* is added to the buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. One hundred bytes beginning at *buffer*+20 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Data Message transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be in the range of 1 to 255.
19	DISK BLOCK IS INTERLOCKED Either the remote system has inhibited transmission on this link because of a backpressure condition or there are no outstanding requests for segments or logical messages from the remote program (assuming it requested segment or message flow control for the link). The calling program is notified with a Link Service Message when the condition clears (see Chapter 4).

Possible Errors (Cont.)

errval	Meaning
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example sends a Network Data Message over a logical link. The ULA is 20. The value for DMFLGS identifies the message data as the sole segment of a logical message and requests the return of link status information. The buffer for returned data is set up as a 40-byte LOGICAL*1 array. Then, after the NTDM call, the returned data is placed as positive integer values in array IR2.

The call sends 400 bytes of data from array A (each real value is 4 bytes long). After the call, the program checks for an error. If none occurred, the program then determines if there are any pending messages. If so, the program branches to receive and process messages. If not, the program continues. For example:

```
      LOGICAL*1  RETBUF(40), R2(40)
      INTEGER  EOM,BOM,RLS,DMFLGS,ERRVAL,ULA,IR2(40),MCNT,MMAX,RIB
      REAL  A(10,10)
      EQUIVALENCE  (IR2(1),R2(1,1)),(MCNT,IR2(32)),(MMAX,IR2(31))
      DO 100 I=1,40
100   R2(2,I) = 0
          .
          .
          .
      EOM = 2
      BOM = 1
      RLS = 128
      ULA = 20
      RIB = 24
      DMFLGS = EOM + BOM + RLS
      CALL NTDM(ERRVAL,RETBUF,ULA,DMFLGS,A,400,0,RIB)
      IF ERRVAL .NE. 0 GOTO 4000
      DO 260 I = 1,40
260   R2(1,I) = R(I)
      IF MCNT .NE. 0 GOTO 3000
```

Send Interrupt Message: NTIN (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message is placed at the head of the pending message queue, behind the first message (since it may already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt Messages are subject to flow control (see Chapter 4).

FORTRAN Call Format

```
CALL NTIN(errval,retbuf,ula,inflgs,buffer,buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTIN call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the starting address of the buffer to receive link status information returned by the call. You need to specify *retbuf* regardless of whether link status information is requested and it has to refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value giving the User Link Address (in the range from 1 to 255) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

The *inflgs* argument is an integer value. It indicates whether link status information is to be returned to *retbuf* when the call completes successfully. This field should be coded as follows:

```
inflgs = rls
```

where:

rls = 128	Link status information is returned.
rls = 0	No link status information is returned.

If link status information is requested, it is returned in the same format as a received Link Service Message. If no link status information is requested, DECnet/E returns random data.

buffer

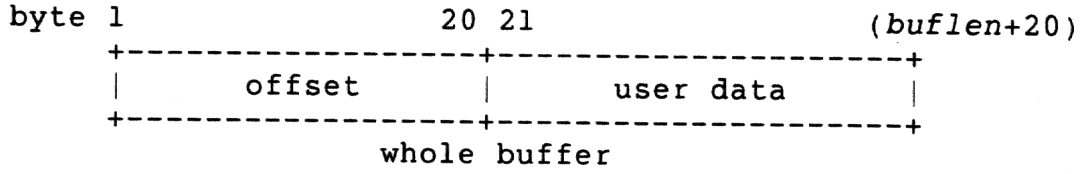
This argument defines the starting address of a buffer containing optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer to determine where the data begins.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	CAN'T FIND FILE OR ACCOUNT The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS() USAGE One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The User Link Address (*ula*) given was zero. Valid values range from 1 to 255.

Possible Errors (Cont.)

errval	Meaning
19	DISK BLOCK IS INTERLOCKED The remote ECL has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message is delivered to the calling program when the condition clears. See Chapter 4 for a full discussion of flow control.
31	ILLEGAL BYTE COUNT FOR I/O The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for an Interrupt Message.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTIN call for logical link 15. The call sends the message TIMEOUT and requests no link status information. A dummy buffer must be defined and referenced, however, so LOGICAL*1 array named DUMMY is defined and used in the call. For example:

```
LOGICAL*1      DUMMY(40)
INTEGER        ERRVAL,INFLGS,ULA,RIB
          .
          .
          .
INFLGS = 0
ULA = 15
RIB = 24
CALL NTIN(ERRVAL,DUMMY,ULA,INFLGS,'TIMEOUT',7,0,RIB)
```

Send Link Service Message: NTLS (DECnet)

The NTLS call:

- Requests data from a remote program over a flow-controlled logical link (see Chapter 4)
- Reenables incoming Interrupt Messages over a logical link (see Chapter 4)
- Obtain status information for the link

No user data is allowed with a Link Service Message.

FORTRAN Call Format

```
CALL NTLS(errval,retbuf,ula,lsflgs,drcnt,ircnt,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLS call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the starting address of the buffer to receive link status information returned by the call. You need to specify *retbuf* regardless of whether link status information is requested and it has to refer to a preallocated buffer at least 40 bytes long.

ula

The *ula* is an integer value identifying the User Link Address (in the range from 1 to 255) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The link service flags argument is an integer value coded as the sum of two values:

lsflgs = rls + opr

where the low-order two bits (*opr*) indicate the purpose of the call:

opr = 0 Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. You can only use this form of the call when the calling program requested message or segment flow control in its Connect Initiate or Connect Confirm Message for this logical link. If no flow control was requested, an NTLS call with *opr* = 0 fails with an error (*errval* = 18).

= 1 Indicates the call is to reenable incoming Interrupt Messages for this link. In this case, the *ircnt* argument is interpreted as the number of Interrupt Messages requested (always 1).

= 2 Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

You can use the *rls* bit when *opr* = 0 or 1 to request that link status information be returned to the return buffer (defined by *retbuf*) when the call completes successfully.

rls = 128 Link status information is returned.
rls = 0 No link status information is returned.

Link status information is always returned when *opr* = 2. If the information is requested, it is returned in the same format as a received Link Service Message. If no link status information is requested, DECnet/E returns random data.

drcnt

The data request count (*drcnt*) is meaningful only when *opr* = 0 and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpreted as either a segment counter (for segment flow control) or a logical message counter (for message flow control). In either case it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local ECL (see Chapter 4). The program returns an error if the *drcnt* specified increases the counter to more than 127.

ircnt

The interrupt request count (*ircnt*) is meaningful only when *opr* = 1 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Hence, the integer value *ircnt* must always be 1 when *opr* = 1 and has the effect of reenabling Interrupt Messages on the logical link. Note that Interrupt Messages can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call fails with an error (*errval* = 19).

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE The Interrupt/Link Service transmit queue for this logical link is full. ECL is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send request after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.

Possible Errors (Cont.)

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PROTECTION VIOLATION Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed causes this error.
18	ILLEGAL SYS( ) USAGE One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *opr* value is not 0, 1, or 2. o The *opr* value is 0, but the program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message. o The data request count (*drcnt*) increased the count of outstanding segments or logical messages requested to more than 127. o The *opr* value is 1, indicating a request for an Interrupt Message, but the *ircnt* argument does not equal 1. o The *opr* value is 1, indicating a request for an Interrupt Message, but the outstanding request counter for Interrupt Messages already equals 1. o The *ula* specified is zero. It must be within the range 1 to 255.

Possible Errors (Cont.)

errval	Meaning
19	DISK BLOCK IS INTERLOCKED The *opr* value is 1, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example of the NTLS call requests five segments from the remote program over logical link 23 on RIB #24. Assume that the program requested segment flow control in its NTCI or NTCC call for this link. For example:

```
INTEGER OPR,RLS,ERRVAL,ULA,LSFLGS,DRCNT,RIB
LOGICAL*1 A(40)
        .
        .
        .
OPR = 0
RLS = 128
ULA = 23
RIB = 24
LSFLGS = RLS + OPR
DRCNT = 5
CALL NTLS(ERRVAL,A,ULA,LSFLGS,DRCNT,0,RIB)
```

Send Disconnect Message: NTDI (DECnet)

The NTDI call terminates an established logical link. Messages in the pending message queue remain but no more messages can be sent over the link. The User Link Address (ULA) is freed and can be used for another logical link. If the last Network Data Message sent did not have the end of message (EOM) flag set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call fails with an error.

Successful completion of an NTDI call assures the calling program that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. However, it does not guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see the section "Send/Receive Disconnect Message" in Chapter 3). Otherwise, the NTDI call provides no particular advantage over an NTLA call in terminating a logical link.

Up to 16 bytes of user data can be sent with the Disconnect Message.

FORTRAN Call Format

```
CALL NTDI(errval,ula,buffer,buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTDI call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value identifying the User Link Address (in the range from 1 to 255) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

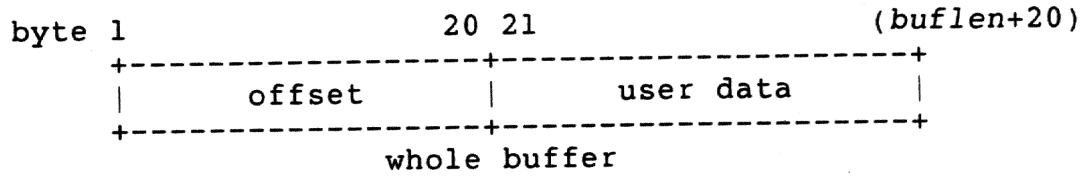
This argument defines the starting address of a buffer containing the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address of the buffer.

buflen

This integer argument defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to define the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a buffer 20 bytes long, an offset of 10, and a *buflen* of 16. In that case, 16 bytes would be sent; that is, the last 10 bytes of the buffer and 6 bytes of whatever is beyond the buffer in memory.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
4	NO ROOM FOR USER ON DEVICE

There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)

5 CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

10 PROTECTION VIOLATION

Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed causes this error.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be within the range 1 to 255.

19 DISK BLOCK IS INTERLOCKED

The last segment sent by the local program over this logical link did not have the *eom* flag set. ECL does not permit a logical link to be disconnected unless all messages have been completely sent. Note that this test of the *eom* flag is done irregardless of the flow control option chosen by the local program.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.

Possible Errors (Cont.)

errval	Meaning
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A retry may succeed.
62	NO RUN-TIME SYSTEM DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	MISSING SPECIAL FEATURE DECnet/E was not installed. The network functions cannot be executed.

Example

The following example disconnects logical link 255 on RIB #24 and includes user data explaining the reason for disconnecting the link. Note that he remote program would have to be coded to recognize this message and take appropriate action. For example:

```
INTEGER ERRVAL,RIB
        .
        .
        .
RIB = 24
CALL NTDI(ERRVAL,255,'NOMORE',6,0,RIB)
```

Send Link Abort Message: NTLA (DECnet)

The NTLA call terminates a logical link immediately. If there are still messages in the transmit queues waiting for acknowledgment from the remote ECL, DECnet/E discards them. These messages may or may not have been received by the remote system (see the section "Send/Receive Link Abort Message" in Chapter 3).

You can send up to 16 bytes of user data with the message. If you issue the NTLA call for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If you issue the NTLA call for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program.

FORTRAN Call Format

```
CALL NTLA(errval,ula,buffer,buflen,bufoff,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the NTLA call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

ula

The *ula* is an integer value identifying the User Link Address (in the range from 1 to 255) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

buffer

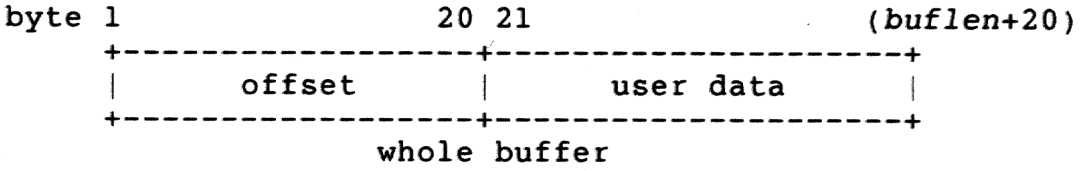
This argument defines the starting address of a buffer containing optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the buffer argument.

buflen

This integer value defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0 to 16.

bufoff

This integer value defines an offset, in bytes, from the address specified by the *buffer* argument. The value of *bufoff* is added to the starting address of the buffer to find the beginning address of the user message data. For example, if *bufoff* = 20, the user data begins in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *buffer*-100 would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT

The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.

18 ILLEGAL SYS() USAGE

One of two possibilities:

- The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list.
- The *ula* specified is zero. It must be in the range of 1 to 255.

31 ILLEGAL BYTE COUNT FOR I/O

The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.

32 NO BUFFER SPACE AVAILABLE

System buffers are not currently available to store this message. A retry may succeed.

62 NO RUN-TIME SYSTEM

DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

66 MISSING SPECIAL FEATURE

DECnet/E was not installed. The network functions cannot be executed.

Example

The following example aborts logical link 17 on RIB #24; no user data is sent:

```
INTEGER RIB
LOGICAL*1  A(256)
         .
         .
         .
RIB = 24
CALL NTLA(ERRVAL,17,A,0,0,RIB)
```

Receive: MRCV (Local and DECnet)

The MRCV call retrieves a message from the calling program's pending message queue. It returns control parameters and user data, if any, to buffers defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer that receives the user data. The message data can be retrieved all at once or in portions, and portions can be discarded. The formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message" show what control information is returned for the various types of messages.

A Receive call can be selected to get:

- The first message in the queue
- The first message in the queue from either a network or a local sender
- The first message in the queue for a particular logical link

You can set a sleep flag and associated timer to indicate that the calling program is to wait if there are no appropriate messages pending in the queue. Otherwise, if there are no appropriate messages pending, the Receive call terminates with an error (*errval* = 5).

FORTRAN Call Format

```
CALL MRCV(errval,retbuf,rmod,sndr,qual,buffer,
          buflen,bufoff,slptime,rib)
```

Argument Descriptions

errval

The *errval* argument must be an integer variable name. This variable is set to zero if the MRCV call succeeds. If an error occurs, *errval* is set to one of the error codes listed under "Possible Errors" at the end of this section.

retbuf

This argument names the starting address of a buffer to contain the control information identifying the type and characteristics of the message received. The predefined buffer must be at least 40 bytes long. The format of the data returned depends on the message type, as described in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

rmod

The receive modifier (*rmod*) argument is an integer value consisting of four flags; *s* + *t* + *l* + *n*, described as follows:

s The sleep flag (*s*) defines what is to be done if there is no appropriate message in the queue for the Receive call to deliver; that is, a message of the type requested with the *l*, *n*, *sndr*, and *qual* arguments. If *s* = 1, the program sleeps (suspended execution) until something happens. The *slptime* argument regulates the length of time that the program sleeps.

If the *s* flag is set to 0, the program does not sleep. If there are no appropriate messages in the queue, the call terminates immediately with an error (*errval* = 5).

t The truncate flag (*t*) is set to 2 to discard any excess information that does not fit in the buffer specified in this call. Information could be left over if the value of *buflen* is smaller than the amount of user data in the message.

A 0 value for *t* indicates that such excess information is kept for retrieval on later Receive calls.

l The local sender flag (*l*) is set to 4 to select a message from a local sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *l* = 0, local selection is not requested. If both *l* and *n* are zero, the first message in the queue is returned.

n The network sender flag (*n*) is set to 8 to select a message from a network sender. Use the *sndr* and *qual* arguments to further qualify the selection.

If *n* = 0, network selection is not requested. If both *l* and *n* = 0, the first message pending in the queue is returned. If *l* = 4, and *n* = 8, local selection prevails.

sndr

The sender selection argument (*sndr*) is an integer value that selects a particular local sender or a particular logical link for this Receive call.

If *l* = 4 (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If *l* = 0 and *n* = 8 (network selection), a nonzero value for *sndr* is interpreted as a ULA. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either *l* = 4 or *n* = 8, the *qual* argument has special meaning. The *sndr* and *qual* arguments are ignored if both *l* and *n* are zero.

qual

The sender selection qualifier (*qual*) is an integer value -- normally zero for user applications.

If *l* = 4 (local selection) and *sndr* = 0, then any nonzero value for *qual* is a special Receive call, requesting a message from the system (job number 0). The RSTS/E utility ERRCPY uses this special Receive call to process messages from the monitor error logging routines.

The *qual* argument is ignored if both *l* and *n* = 0 or if *sndr* $\neq$ 0.

Table 10-4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 10-4: Sender Selection Summary

rmod				
n	*l*	*sndr*	*qual*	Result
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	4	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times two. Only messages from that job are delivered on this MRCV call.
8	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a ULA. Only network messages from the designated logical link are delivered on this MRCV call.

buffer

This argument defines the starting address of the buffer to which user message data is to be delivered. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by the *buffer* argument.

buflen

This integer argument defines the maximum number of bytes to be delivered to the buffer on this Receive call.

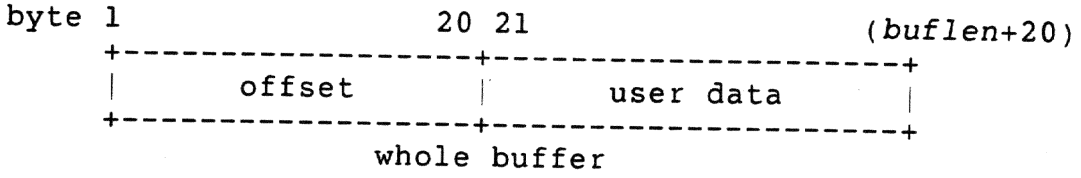
If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (*t* flag of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, only the amount indicated by *buflen* is returned and the rest is saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining bytes 9-10 in the data returned to the *retbuf* buffer by the Receive call.

The number of bytes actually returned to the buffer is returned in bytes 13-14 in the *retbuf* buffer. See the formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

bufoff

This integer value defines an offset, in bytes, from the beginning of the buffer specified by the buffer argument. The value of *bufoff* is added to the starting buffer address to define the beginning address of the user message data. For example, if *bufoff* = 20, the user message data is delivered beginning in byte 21 of the buffer.



Note

The FORTRAN interface makes no checks for inconsistencies in the *buffer*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. Up to 100 bytes beginning at *buffer*+20 could be delivered on the Receive call.

slptime

The *slptime* argument, an integer value, is significant only when the *s* flag of *rmod* = 1. In that case, *slptime* defines the length of time (in seconds), that a program is to sleep when no appropriate message is in the queue. The program sleeps until:

- The sleep timer expires.
- Any new message is placed in the queue (not just an appropriate one).
- A delimiter is typed on a terminal opened by, or assigned to, the job.
- Log-ins are disabled. (This could occur if the system is being shut down).
- A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
- The job itself has opened the XM: (DMC11 driver) and a message is received for the job.

In all cases, the program is awakened with an error (*errval* = 5) but is not passed a message. To obtain a pending message, the program must execute another Receive call. Since the program can be awakened by any of the conditions listed previously, you can check for pending messages by executing a MRCV call without a sleep.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors

errval	Meaning
5	CAN'T FIND FILE OR ACCOUNT For a Receive call without sleep (*s* flag in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (*s* flag in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV call to retrieve any pending messages.
18	ILLEGAL SYS() USAGE No RIB exists for the calling program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example

In the following example, the first Receive call (on RIB #24) is issued for Network Data Messages with the sleep timer set to five seconds. The truncate bit and length argument are 0, so all available information is saved for the second Receive call. The test for ERRVAL = 5 after the first Receive call (with sleep) ensures that a message is waiting when the second Receive call is issued. The computed GO TO checks byte 3 of the returned data, identifies the type of message, and branches accordingly. This byte is a signed value, so it is not necessary to convert it to an integer.

```
      LOGICAL*1  RETBUF(40),BUFFER(256)
      INTEGER S,N,ERRVAL,RMOD,SLPTIM,RIB
         .
         .
         .
      S = 1
      N = 8
      ERRVAL = 0
      RMOD = S + N
      SLPTIM = 5
      RIB = 24
110   CALL MRCV(ERRVAL,RETBUF,RMOD,0,0,BUFFER,0,0,SLPTIM,RIB)
      IF ERRVAL .EQ. 5 GOTO 110
      RMOD = N
120   CALL MRCV(ERRVAL,RETBUF,RMOD,0,0,BUFFER,256,0,0,RIB)
      IF ERRVAL .NE. 0 GOTO 4000
130   GO TO (200,300,400,500,600,700,800,900,1000) RETBUF(3)+10
200   (processing for Link Abort Message)
300   (processing for Disconnect Message)
            .
            .
            .
```

Format of Received Local Data Message

The following format describes the information returned to the *retbuf* buffer for a Local Data Message. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive (MRCV) call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -1, function code for a Local Data Message.
4	byte	The job number times two of the local sender.
5-6	2 bytes	The project-programmer number of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-40	(however sent)	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 20 bytes.

Format of Received Local Data Message with Privilege Mask

The following format describes the information returned to the *retbuf* buffer for a Local Data Message with Privilege Mask. Up to 512 bytes of user data can be delivered to the data buffer specified in the Receive (MRCV) call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -11, function code for a Local Data Message with Privilege Mask.
4	byte	The job number times two of the local sender.
5-6	2 bytes	The project-programmer number of the local sender. Byte 5 = programmer number, byte 6 = project number.
7	byte	Keyboard number of the sender, or -1 if the sender is detached.
8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the buffer on this Receive call.
15-20	--	Reserved.
21-28	8 bytes	The sender's privilege mask.
29-40	(however sent)	The data passed as parameters by the sender of this message. The system pads any unused bytes with binary zeros to a length of 12 bytes.

Format of Received Connect Initiate Message

The following format describes the information returned to the *retbuf* buffer for a received Connect Initiate Message. A 120-byte Connect Data Block and up to 16 bytes of user data are also returned to the data buffer specified in the Receive call.

Figure 10-2 shows the format of a received Connect Data Block. Table 10-5 describes the format in detail.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -2, function code for a Connect Initiate Message.
4	--	Reserved.
5-6	integer	Local Link Address. Identifies the link in the local program's subsequent Connect Conform or Connect Reject Message.
7-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the buffer on this Receive call.
15-22	--	Reserved.
23	byte	Remote link modifier. Indicates the flow control in effect for the remote program. The low-order two bits should be masked and interpreted as follows:

```
LOGICAL*1    RETBUF(40)
INTEGER      RLMOD
   .
   .
   .
RLMOD = RETBUF(23)
FLOW = RLMOD .AND. 3
```

Format of Received Connect Initiate Message (Cont.)

Bytes	Data Type	Description
		Then, FLOW = 0 No flow control = 1 Segment flow control = 2 Message flow control When the byte value RETBUF(23) is stored in the integer variable RLMOD, the sign extension is insignificant since we are masking off all but the two low-order bits.
24	--	Reserved.
25-26	integer	Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. This value is supplied by the local ECL and is determined by the size of the receive buffers that it allocates. The maximum size of the receive buffers is set for the ECL by Network Management. The program can further limit the amount of user data to be received by specifying a smaller *rmax* in the Connect Confirm Message that accepts this request for a logical link (see the section "Send Connect Confirm Message").
27-28	integer	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

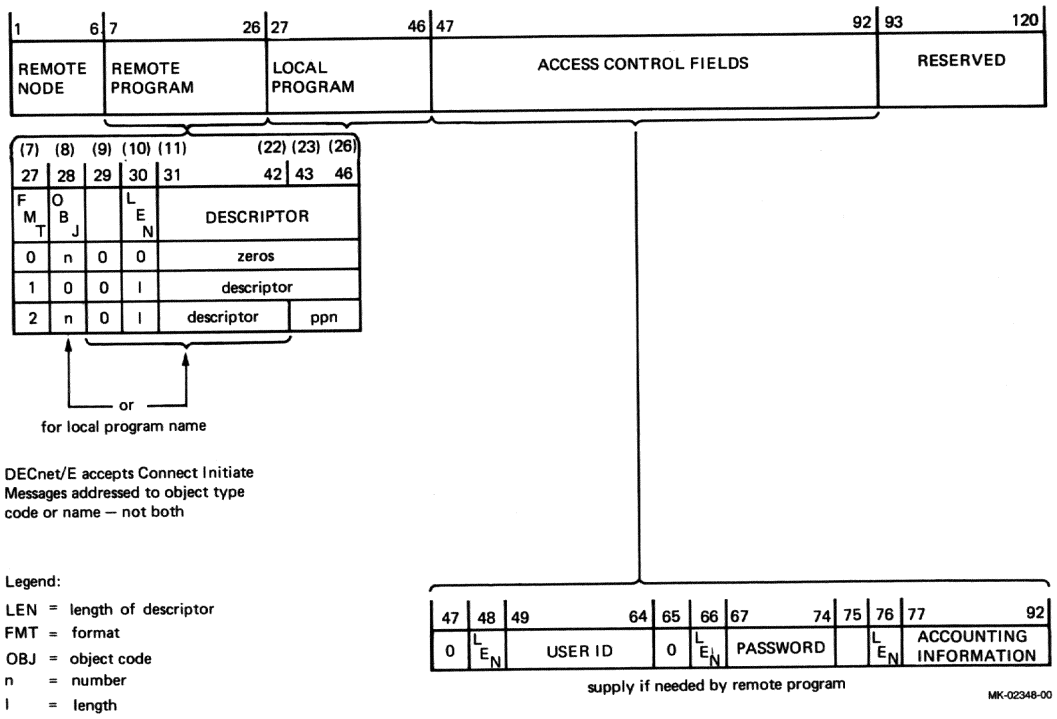


Figure 10-2: Format of Received Connect Data Block

Table 10-5: Format of Received Connect Data Block

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node from which the Connect Initiate Message was received. Names shorter than 6 characters are left-justified with space fill to 6 characters.
7-26	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of the following three formats.

Table 10-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 0 The remote program is identified by object type code alone. The fields are: **Bytes** — **Content** 7 — Zero, to indicate a format 0 name. 8 — Object type code of the remote program. 9-26 — Zero (not used).
	Format 1 The remote program is identified by a descriptor. The subfields are: **Bytes** — **Content** 7 — One, to indicate a format 1. 8 — Normally zero. This byte is usually zero unless the target DECnet system identifies the remote program with both name and object type code on outgoing connect requests. 9 — Reserved. 10 — Length (in bytes) of the descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26). 11-26 — Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Table 10-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 2 This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC. DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved.
10	Length (in bytes) of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor is left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E PPN.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E PPN.

Table 10-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
27-46	**Local Program Name.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Equal to 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.

Table 10-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
47-92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most precise checking done in a normal login on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system passes this information on to the receiving program. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49-64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN at login.
65	Reserved.
66	Length of password information specified in the next field, 0 to 8 bytes.
67-74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login.

Table 10-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	75 Reserved.
	76 Length of the accounting information specified in the following field, 0 to 16 bytes.
	77-92 Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.
93-120	Reserved.

Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the local program. The link is identified by the User Link Address (ULA) specified by the local program in the Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. You can also return up to 16 bytes of user data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -3, function code for a Connect Confirm Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate Message.
5-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23	byte	Remote link modifier. Indicates the flow control in effect for the remote program. The low-order two bits should be masked and interpretted as follows:

```
LOGICAL*1    RETBUF(40)
INTEGER      RLMOD,FLOW
   .
   .
   .
RLMOD = RETBUF(23)
FLOW = RLMOD .AND. 3

Then, FLOW = 0  No flow control
           = 1  Segment flow control
           = 2  Message flow control
```

Format of Received Connect Confirm Message (Cont.)

Bytes	Data Type	Description
24	--	Reserved.
25-26	integer	Receive maximum, in bytes. The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This is: o Equal to the *rmax* value specified by the local program in the Connect Initiate Message for this link o A lesser value supplied by the local ECL, determined from the size of the receive buffers it allocates.
27-28	integer	Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.
29-40	--	Reserved.

Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the local program's request for a logical link. The link is identified by the User Link Address (ULA) previously established in the local program's Connect Initiate Message. The following format describes the information returned to the *retbuf* buffer. Rejections from ECL contain a reason code explaining the reason for the rejection and do not contain user data. Rejections from the remote program can contain up to 16 bytes of user data, delivered to the data buffer defined in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -4, function code for a Connect Reject Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate Message.
5-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23-24	integer	Identifies the reason for the rejection (see Appendix B).
25-40	--	Reserved.

Format of Received Network Data Message

A received Network Data Message contains user data from a remote program. The following format describes the information returned to the *retbuf* buffer. The user data accompanying the Network Data Message is delivered to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -5, function code for a Network Data Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-20	--	Reserved.
21	byte	Data message flags. The low-order bits indicate whether this is the beginning, middle, end, or sole segment of a logical message. These bits should be masked and interpreted as follows:

```
BYTE    RETBUF(40)
INTEGER DMFLGS,BOM,EOM
  .
  .
  .
DMFLGS = RETBUF(21)
BOM = DMFLGS .AND. 1
EOM = DMFLGS .AND. 2
```

Format of Received Network Data Message (Cont.)

Bytes	Data Type	Description

where:

eom	*bom*	Meaning
0	0	Middle segment of message
0	1	First segment of message
2	0	Last segment of message
2	1	Sole segment of message

If the local program requested message flow control when the logical link was established, DECnet/E uses the EOM bit to determine the end of a logical message. It does not use the BOM bit but passes it on to the receiving program.

22-40	--	Reserved.

Format of Received Interrupt Message

The local ECL places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The following format describes the data returned to the *retbuf* buffer. You can deliver up to 16 bytes of user data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -6, function code for an Interrupt Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the *t* flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-40	--	Reserved.

Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call when the pending message queue has emptied after a condition has cleared that previously inhibited the local program from sending (see the section "Flow Control" in Chapter 4). Status information with the same format as a Link Service Message can also be returned to the *retbuf* buffer on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The following format describes the information returned. No data is delivered to the data buffer with a Link Service Message.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -7, function code for a Link Service Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	integer	Always zero for Link Service. (These bytes are the remainder field in other returned messages.)
11-12	--	Reserved.
13-14	integer	Always zero for Link Service. (These bytes are the length field in other returned messages.)
15-20	--	Reserved.
21	byte	Local status flags. These flags should be masked and interpreted as follows:

```
LOGICAL*1  RETBUF(40)
INTEGER    LSTAT,DATA,INTLS,LCLBP
  .
  .
  .
LSTAT = RETBUF(21)
DATA = LSTAT .AND. 1
INTLS = LSTAT .AND. 2
LCLBP = LSTAT .AND. 192
```

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
		If DATA = 1, outgoing Network Data Messages are inhibited. (A Link Service Message is delivered when the condition clears.) If INTLS = 2, outgoing Interrupt and Link Service Messages are inhibited. (A Link Service Message is delivered when the condition clears.) The LCLBP value indicates the local NSP's backpressure status for this link: **LCLBP — Meaning** 0 — Incoming Data Message flow is on. 64 — Incoming Data Message flow is on; however, the local ECL turns this link off if another Data Message is received before the local program's receive queue is emptied. 128 — Incoming Data Message flow has been turned off by the local ECL. The local program's receive queue must be emptied before the link is turned on. 192 — Incoming Data Message flow has been turned off by the local ECL but flow is scheduled to turn on as soon as system buffers become available.

22	byte	Remote status flags. Only one bit is currently defined for this field. This bit should be masked and interpreted as follows:
		
		`LOGICAL*1  RETBUF(40)`
		`INTEGER  REMBP,RSTAT`
		`.`
		`.`
		`.`
		`RSTAT = RETBUF(22)`
		`REMBP = RSTAT .AND. 128`
		
		If REMBP = 128, the link has been turned off by the remote system due to a backpressure condition. No Data Messages can be sent until the remote system turns the link back on.

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
23	byte	Local data request count. The count of segments or logical messages currently requested by the local program, reflecting the actual number of segments or logical messages requested but not yet received.
24	byte	Local interrupt request count. The count of Interrupt Messages currently requested by the local program, reflecting the actual number of Interrupt Messages requested but not yet received. This count never exceeds one.
25	byte	Remote data request count. The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or messages requested but not yet sent.
26	byte	Remote interrupt request count. The count of Interrupt Messages currently requested by the remote end of the logical link, reflecting the actual count of Interrupt Messages requested but not yet sent.
27	byte	Data transmit queue maximum. The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.
28	byte	Data transmit queue count. The number of Data Messages currently in the data transmit queue for this local link. It is a count of data messages waiting for acknowledgment from the remote ECL.
29	byte	Interrupt/Link Service transmit queue maximum. The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.

Format of Received Link Service Message (Cont.)

Bytes	Data Type	Description
30	byte	Interrupt/Link Service transmit queue count. The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote ECL.
31	byte	Message maximum. The maximum number of incoming messages that are queued for the local program. This maximum is set by the program in its Declare Receiver call.
32	byte	Message count. The total number of messages currently in the pending message queue for the local program.
33	byte	Message count for this logical link. The number of messages currently in the pending message queue for the logical link identified by the *ula* in byte 4.
34-40	--	Reserved.

Format of Received Disconnect Message

A received Disconnect Message indicates that an established logical link has been disconnected by the remote program. All other messages sent by the remote program over this logical link have been received by the local program. The following format describes the data returned to the *retbuf* buffer. Up to 16 bytes of user message data can be returned to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -8, function code for a Disconnect Message.
4	byte	User Link Address, previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-40	--	Reserved.

Format of Received Link Abort Message

A received Link Abort Message indicates that a logical link has been terminated. The following format describes the data returned to the *retbuf* buffer. You can return up to 16 bytes of user message data to the data buffer specified in the Receive call.

Bytes	Data Type	Description
1-2	--	Reserved.
3	byte	The value -9, function code for a Link Abort Message.
4	byte	User Link Address (if any), previously defined in the local program's Connect Initiate or Connect Confirm Message.
5-6	integer	Local Link Address, established by the local ECL for this link. It identifies the link if the remote program has sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No *ula* exists under these conditions to identify the link being aborted.
7-8	--	Reserved.
9-10	integer	The number of bytes of user data not delivered to the data buffer. This data has been either discarded or saved for a later Receive call, depending on how the t flag was set in the *rmod* argument.
11-12	--	Reserved.
13-14	integer	The number of bytes of user data transferred to the data buffer on this Receive call.
15-22	--	Reserved.
23-24	integer	Abort reason. If this is nonzero, the link was aborted by either the local or remote ECL and the reason is specified in these two bytes. See Appendix B for a list of reasons that apply for a link abort. If reason is zero, the link was aborted at the request of the remote program.
25-40	--	Reserved.

Chapter 11

Network Programming in MACRO

Programming Background

This chapter presents the specific calls, with detailed formats, used to access the DECnet/E message services in a MACRO assembly language program. These calls are expanded at assembly time from a library file. The library file must be included with your source files for the assembly to produce an object module with the expansions. The expansions, in turn, call subroutines from another library file that must be included with your assembled object file at link time to produce an executable program. Thus, there are three aspects to using the message calls in a MACRO program: coding, assembling, and linking.

Coding

The message macro expansions use internal macros that must be defined in your program. The initialization macro .MSR (without any arguments) defines the necessary internal macros and must itself be established (with .MCALL) and invoked before any of the message calls are used. For example:

```
.mcall   .msr
.msr
```

The message calls used in your program must also be declared in an .MCALL directive; for example,

```
.mcall $mdcl,$mrem,.ntci,.ntcr,.ntls,.ntla,.mrcv
```

This example shows that there are two forms for each message call: the "." form and the "$" form.

"." Form of Message Calls

The . form (for example, .NTCI) expands to code that:

- Inserts values that you specify into a 10-word argument block at execution time. You must allocate space for the argument block in a data section of your program. Only the arguments specified in the call are stored in the block. Any unused or unspecified arguments in the argument block are left untouched -- denoted throughout this chapter with slashes (///////). The arguments are stored in the block at execution time. Thus, the arguments you specify in the call should be appropriate as operands in MOV instructions.

- Moves the address of the argument block into R0. (You must save R0 before each message call, if necessary, and restore it later. R0 is set with an error code after each call.)

- Calls a subroutine named $$$MSR. This subroutine examines the argument block to determine which of the message operations is to be executed. It then sets up the job's File Request Queue Block (FIRQB) and Transmit Request Block (XRB) and executes a .MESAG monitor directive. This directive then transfers control to the appropriate section of the DECnet/E code in the RSTS/E monitor.

  (The FIRQB and XRB are two areas within the first 512. bytes of virtual address space reserved for communications between the RSTS/E monitor, the run-time system, and the user job. See the *RSTS/E System Directives Manual* for general description of the first 512. bytes of virtual memory, along with descriptions of the FIRQB and the XRB.)

  DECnet/E performs the function requested, if possible, and transfers control back to the user program at the location following the call. If the call succeeds, the carry bit is clear and R0 is set to zero. If the call fails, the carry bit is set and R0 is set to an error code. The program can test R0 for numeric values or use error mnemonics resolved from the file ERR.STB at link time (see the section "Linking").

Note

Chapter 12 contains the FIRQB and XRB layouts for each of the message calls. These layouts are provided for program debugging only. You should not attempt to set up the FIRQB and XRB and invoke the .MESAG directive yourself. DECnet message operations should be accessed only with the calls described in this chapter.

For example, consider a call, .DECNTE, whose calling sequence is:

```
.DECNTE   area,arg1,arg2,arg3,arg4
```

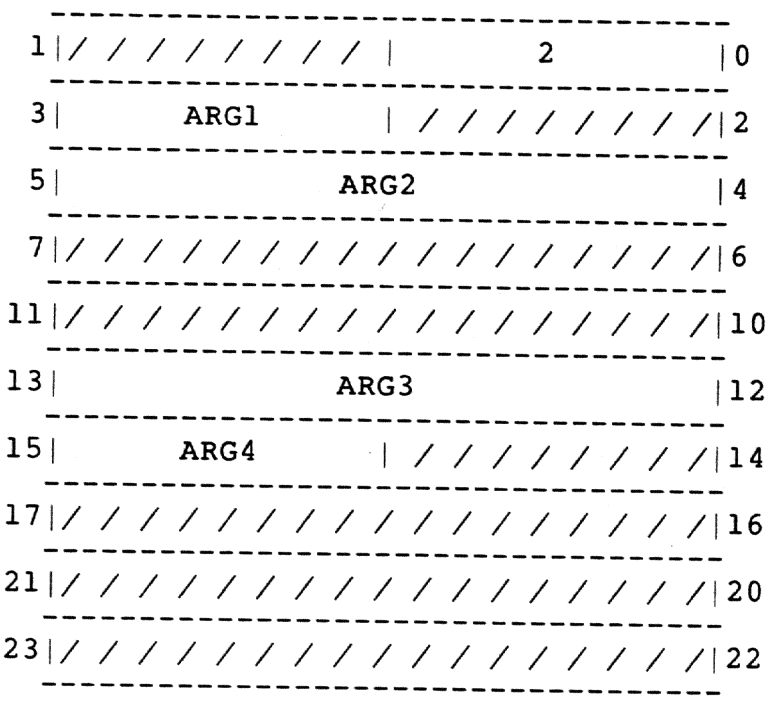
A program might call the macro with code as follows:

```
        .MCALL  .DECNTE
AREA:   .WORD   0,0,0,0,0,0,0,0,0,0     ;DEFINE ARGUMENT BLOCK
            .
            .
            .
        .DECNTE #AREA,#ARG1,#ARG2,#ARG3,#ARG4
```

After assembly, the code inserted for the .DECNTE macro is similar to:

```
MOV             #AREA,R0
MOVB            #2,(R0)
MOVB            #ARG1,3(R0)
MOV             #ARG2,4(R0)
MOV             #ARG3,10.(R0)
MOVB            #ARG4,13.(R0)
JSR             PC,$$$MSR
```

Thus, at execution time, the argument block is filled as follows:



The JSR instruction transfers control to the $$$MSR subroutine that examines the argument block. The subroutine reads the first byte and then branches to the section of code that handles the specified operation.

"$" Form of Message Calls

The "$" form of the message calls (for example, $MDCL) expands to code that defines an argument block filled with argument values. The values are filled in at assembly time. Thus, the arguments you specify in the call should be appropriate as operands in MACRO-11 directives such as .BYTE, .WORD, and so forth. Unlike the "." form of the call, any unused or unspecified areas in the argument block are filled with zeros. Since the "$" form of the call only allocates and fills the argument block, such calls should be in a data section of the program. To execute the call, the program must load R0 with the address of the argument block, and execute the call to the $$$MSR subroutine.

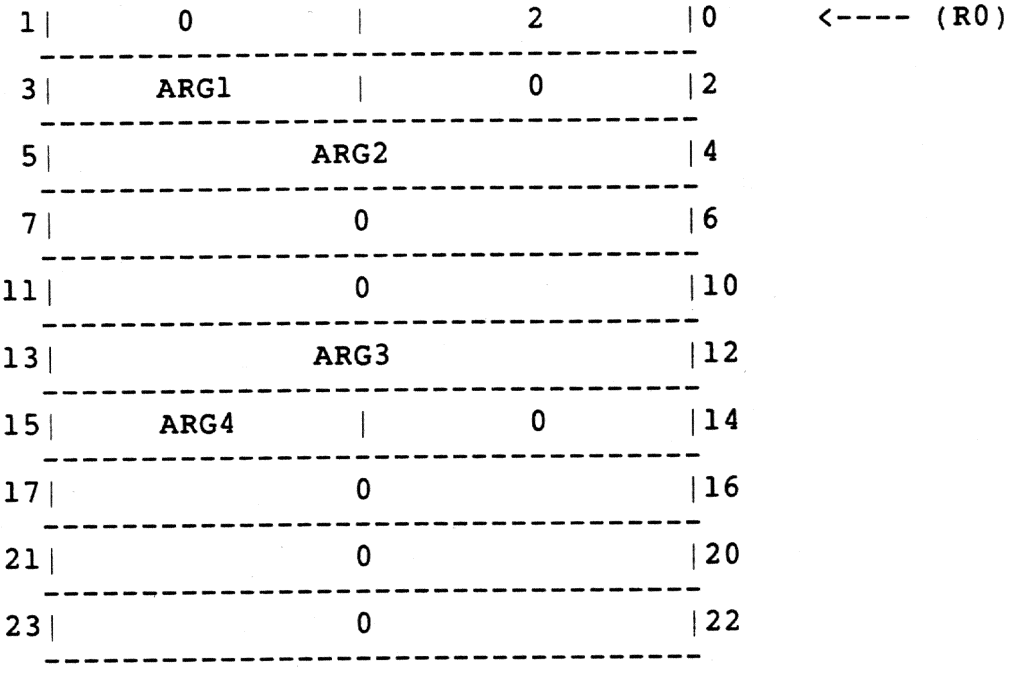
The "$" form of the sample macro discussed previously would then be coded as follows:

```
        .MCALL    $DECNTE
  AREA: $DECNTE   ARG1,ARG2,ARG3,ARG4
          .
          .
          .
        MOV       #AREA,R0
        JSR       PC,$$$MSR
```

After assembly, the code inserted for the $DECNTE macro is:

```
  AREA:  .BYTE    2,0,0
         .BYTE    ARG1
         .WORD    ARG2
         .WORD    0, 0
         .WORD    ARG3
         .BYTE    0,ARG4
         .WORD    0,0,0
```

And the argument block created the assembler is:



The expansions for the actual message calls are not shown in this chapter. The calls could change or be enhanced in later versions of DECnet/E, and expanding the calls yourself could lead to extensive modifications later. However, you can use the MACRO-11 .LIST directive to list the expansions in the program.

Assembling

You can assemble MACRO-11 source code that includes DECnet subroutine calls with either one of the two MACRO-11 assemblers available on RSTS/E systems:

- The MAC assembler (for code to be run under the RSX Run-Time System or its derivatives)
- The MACRO assembler (for code to be run under the RT11 Run-Time System)

The library file containing the expansions for the MAC assembler is LB:NETMLB.MLB. This file must be included in the source file list (on the right side of the equal sign) in the assembly. See the *RSTS/E Programmer's Utilities Manual* for complete instructions on how to assemble programs using the MAC assembler. A typical sequence for an assembly is:

```
$ RUN $MAC
MAC>objfile,lstfile=LB:NETMLB.MLB/ML,srcfill,srcfil2
MAC>^Z

$
```

The library file containing the expansions for the MACRO assembler is LB:NETMLB.SML. This file must be included in the source file list (on the right side of the equal sign) in the assembly. See the *RSTS/E RT11 Utilities Manual* for complete instructions on how to assemble programs using the MACRO assembler. A typical sequence for an assembly with MACRO is:

```
$ RUN $MACRO
*objfile,lstfile,crefile=LB:NETMLB.SML/M,srcfill,srcfil2
*^Z

$
```

Another useful file to include in the assembly source file list for either MAC or MACRO assemblies is the file COMMON.MAC. DECnet/E calls use the FIRQB and the XRB to return data to the user program. COMMON.MAC assigns the mnemonics FIRQB and XRB to the address of the first byte of the appropriate areas and also contains useful macros. One in particular, .BLKW0, is useful in conjunction with the message calls. Like the MACRO-11 .BLKW directive, .BLKW0 allocates a specified number of words of storage. In addition, .BLKW0 fills the contents of the locations allocated with zeros. For example:

```
CDB:  .BLKW0 172.
```

This statement allocates 172 zero-filled words.

Linking

As with assemblers, two linkers are available on RSTS/E systems. Use the Task Builder (TKB) to link object modules produced by the MAC assembler (RSX Run-Time System and its derivatives). The Task Builder automatically resolves references to the message subroutines from the system library (LB:SYSLIB.OLB). See the *RSTS/E Task Builder Reference Manual* for a detailed description of the Task Builder. A typical link operation with TKB is:

```
$ RUN $TKB
TKB>binfile,mapfile=objfile1,objfile2,...
TKB>//
TKB>^Z

$
```

Use the LINK routine to link object modules produced by the MACRO assembler (RT11 Run-Time System). LINK resolves references to the message subroutines from the system library ($SYSLIB.OBJ). See the *RSTS/E RT11 Utilities Manual* for a detailed description of LINK. A typical link operation with LINK is:

```
$ RUN $LINK
*binfile,mapfile=objfile1,objfile2,...
*^Z

$
```

Another useful file that can be included in a link operation is the file ERR.STB. ERR.STB defines mnemonics for the numeric error codes returned in R0 by the message calls. The descriptions in this chapter list the error mnemonics provided by ERR.STB, as well as the octal error values. LINK automatically resolves the symbols at link time; you do not need to include ERR.STB in the source file list.

Declare Receiver: MDCL (Local and DECnet)

You must execute the MDCL call before a program can receive any messages or send any network messages. However, local messages can be sent without first executing this call. The call defines an identifying object type code or name (or both) and any restrictions on incoming messages. The monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block (RIB) for the job (see the section "DECnet/E Use of the Declare Receiver Call" in Chapter 3). Only one MDCL call can be in effect at a time for a particular job. It remains in effect until you execute a Remove Receiver (MREM) call for the job.

MACRO Format

```
.MDCL area,namptr,objtyp,access,#0,mmax,lmax,rib
```

or

```
label:  $MDCL namptr,objtyp,access,0,mmax,lmax,RIB
```

Argument Descriptions

label

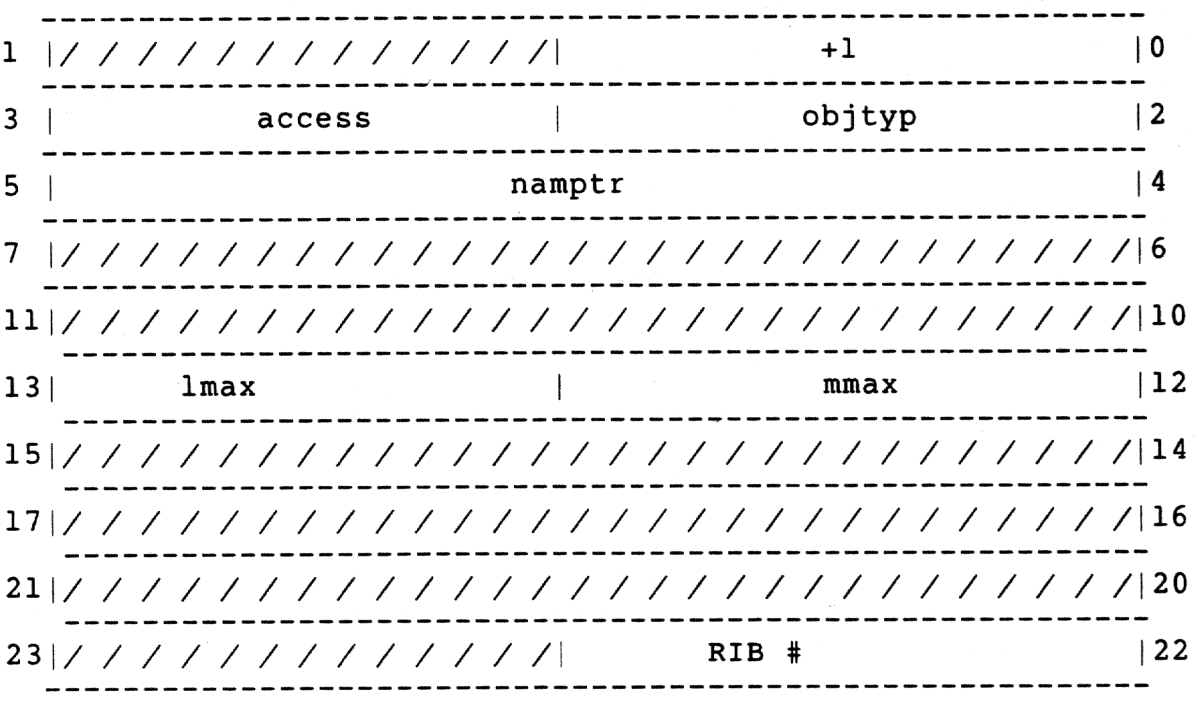
The label used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

> The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MDCL call, the argument block contains:



namptr

This word is the address of a six-byte area containing the receiver's logical name. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). If the name is less than six characters, it must be left-justified in the 6-byte area and the remaining bytes must be filled with spaces. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), this area should contain six spaces.

The name identifies the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code, if one is declared.

For local or network programs to access the calling program by logical name, the name must be nonnull and unique to the node. That is, only one program at the node can declare its identity with the name at any given time. The name is used to identify the calling program for queuing messages from local or network programs. If the logical name is null, only network access by object type code is allowed (see the discussion of *namptr*, *objtyp*, and *access*).

objtyp

This byte value specifies the object type code, another form of network addressing. As the section "Declaring Identity" in Chapter 3 discusses, the object type code defines some service the calling program performs. If the program that issues the MDCL call is addressed by object type code (*namptr* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environment. Unlike the logical name, the object code need not be unique within the node -- multiple copies of a program can declare their identity with the same type code simultaneously. If *namptr* is null, *objtyp* cannot be zero. Chapter 4 discusses network addressing in detail.

Acceptable values for the object type code range from 0 through 255 (0 to 377 octal). The value 0 indicates that other programs in the network never addresses the local program by object type code. The range from 1 to 127 (1 to 177 octal) is reserved for DIGITAL use (see Appendix A). The range from 128 to 255 (200 to 377 octal) is available for definition and use by a network installation.

access

This byte contains bit flags that are used to determine access to the calling program and to modify certain aspects of the message operations.

The low-order three bits limit the types of senders that can queue messages for the calling program. However, they do not limit the messages that the calling program can send.

bit 0 = 0 Local senders cannot queue messages for the calling program. (Local senders who use the network functions are considered network senders in this context.)

= 1 Local senders can queue messages for the calling program.

bit 1 = 0 Both local senders with and without privilege can queue messages for the calling program.

= 1 Only local senders with SYSIO privilege can queue messages for the calling program.

Note that this bit is ignored if bit 0 = 0 (that is, if no local senders are allowed).

bit 2 = 0 Session Control rejects any incoming requests for logical links.

= 1 ECL queues incoming Connect Initiate Messages for the local program, subject to the declared link maximum (*lmax*) and the declared message maximum (*mmax*). In this case, the user program must provide its own protection from unauthorized access, if desired.

A useful way to document and set these bits is to assign mnemonics to set the bit values:

```
L = 1
P = 2
N = 4
```

The byte value can then be set using a statement of the form ACCESS = L+P+N.

Table 11-1 summarizes access code values and the access permitted for each.

Table 11-1: Types of Access Permitted

Access Code (bits 0,1,2)	Network Logical Links	Local Senders with Privilege	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit 3 of the *access* byte is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If bit 3 = 0, incoming requests for logical links do not affect the link maximum. However, if bit 3 = 1 after queuing an incoming Connect Initiate Message to the receiver, ECL sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 1, a single-link program prevents the possibility that ECL queues a Connect Initiate Message for it after its one logical link is disconnected and before it issues a Remove Receiver call.

The DECnet/E FAL utility, for example, sets the one-shot bit to 1 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote NFT utility, ECL queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that NFT breaks the first link and the time that FAL issues its Remove Receiver call. This makes sure that each incoming connect request starts a new copy of FAL.

Bit 4 of the *access* byte is used to modify the function of the RSTS/E conditional .SLEEP monitor directive. If bit 4 = 0, any unreceived message queued for the program blocks the execution of a conditional Sleep call. This is the normal RSTS/E function. However, if bit 4 = 1 the RSTS/E monitor does not check the program's message queue when determining whether or not it should suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can block the Sleep call, but a pending message does not.

namptr, objtyp,* and *access

ECL checks the logical name (whose address is given in *namptr*), and the *objtyp* and *access* arguments to make sure that they are consistent. For example, a null logical name and an access allowing local senders are inconsistent, since local senders can address the program by logical name. ECL checks these arguments as shown in Table 11-2. If the arguments are invalid, ECL returns an error (BADFUO) in R0. If the arguments are valid, ECL allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (NOROOM). A retry may succeed.

Table 11-2: System Validation of Name, Object, and Access Parameters

Access Code (bits 0,1,2)	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local (only)	Ignored	Must be nonnull and unique.
4,6	Network (only)	Zero	Must be nonnull and unique.
		Nonzero	If *namptr* is null (at least one leading binary zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If *namptr* is nonnull, it must be unique at the local node. Network senders can refer to the calling program by name or by object type. Only one copy of the program using this logical name can execute at any given time.
5,7	Network or Local	Any value	Must be nonnull and unique so local senders can refer to the calling program by logical name. Network senders can use the logical name or object type (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

mmax

The message maximum byte value (*mmax*) limits the number of messages that can be queued for the calling program. The value can range from 0 to 255 (decimal).

ECL keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the maximum set by the receiver:

- Incoming Local Data Messages are no longer queued for the receiving program. ECL returns an error to any local sender that tries to send a message when the receiving program's pending message count equals or exceeds *mmax*.

- Incoming Connect Initiate Messages cause ECL to see if another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, ECL rejects the connection; the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

- An incoming Network Data Message causes NSP to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message, forcing the remote system to retransmit when flow is reenabled. See the section "Backpressure Flow Control" in Chapter 4.

lmax

The link maximum byte (*lmax*) limits the number of incoming requests for logical links for this program. If the number of links currently active is greater than or equal to *lmax* (0 to 63 decimal) when a remote Connect Initiate Message is received for this program, the local Session Control determines if another copy of the program can be started automatically (see the section "Handling of Incoming Connect Requests" in Chapter 4). If not, Session Control rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program avoids the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting bit 2 of the access byte equal to zero. That is, Session Control rejects all incoming requests for logical links to this program. However, the *lmax* argument does not limit the number of logical links initiated by this program. That is, it does not stop the program from sending Connect Initiate Messages.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs), numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call. The system allocates space for each RIB from the small buffer pool.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
3	INUSE	The calling job already exists in the system's list of declared receivers. This error can indicate a logic error in the program or that a previous program, running under the same job number, failed to remove itself from the receiver list before terminating. In the last case, issue a Remove Receiver call followed by another Declare Receiver call. It is common practice to code a Remove Receiver call immediately before the Declare Receiver call.
4	NOROOM	There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.
10	PRVIOL	The calling program must have SYSIO privilege at the time the Declare Receiver call is executed.
16	FIEXST	The logical name passed is being used by another job.
18	BADFUO	The parameters passed are inconsistent. See *logical name*, *object*, and *access* discussion.

Example

The following example shows a Declare Receiver call allowing only network access using logical links. Both a logical name (FENWCK) and an object type code (130 decimal) are declared. Since local messages are not allowed, the buffer maximum is set to zero. The message maximum is set to 5. If more than 5 messages accumulate for this program, an incoming request for a connection causes Session Control to determine if another copy of the program can be started automatically. If not, Session Control rejects the link. Any incoming data messages at this point also cause Session Control to inhibit incoming message flow on all the program's logical links because of the backpressure condition. The link maximum is set to two. When the program has established two logical links, an incoming Connect Initiate Message causes Session Control to check to see if another copy of the program can be started. If not, Session Control rejects the link. RIB #5 is used. For example:

```
AREA:    .BLKW0 10.
LNAME:   .ASCII /FENWCK/
             .
             .
             .
         .MDCL    #AREA,#LNAME,#130.,#4,#0,#5,#2,#5
```

The following example shows the $MDCL form of the same call:

```
LNAME:   .ASCII /FENWCK/
AREA:    $MDCL   LNAME,130.,4,0,5,2,5
             .
             .
             .
         MOV     #AREA,R0
         JSR     PC,$$$MSR
```

Remove Receiver: MREM (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. Session Control discards all pending messages and aborts any active logical links. Execute this call at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Programs with JOBCTL privilege can remove other RSTS/E jobs with this call, although normally only system utility programs would do this.

MACRO Format

```
.MREM area,jobx2,rib
```

or

```
label:  $MREM jobx2,rib
```

Argument Descriptions

label

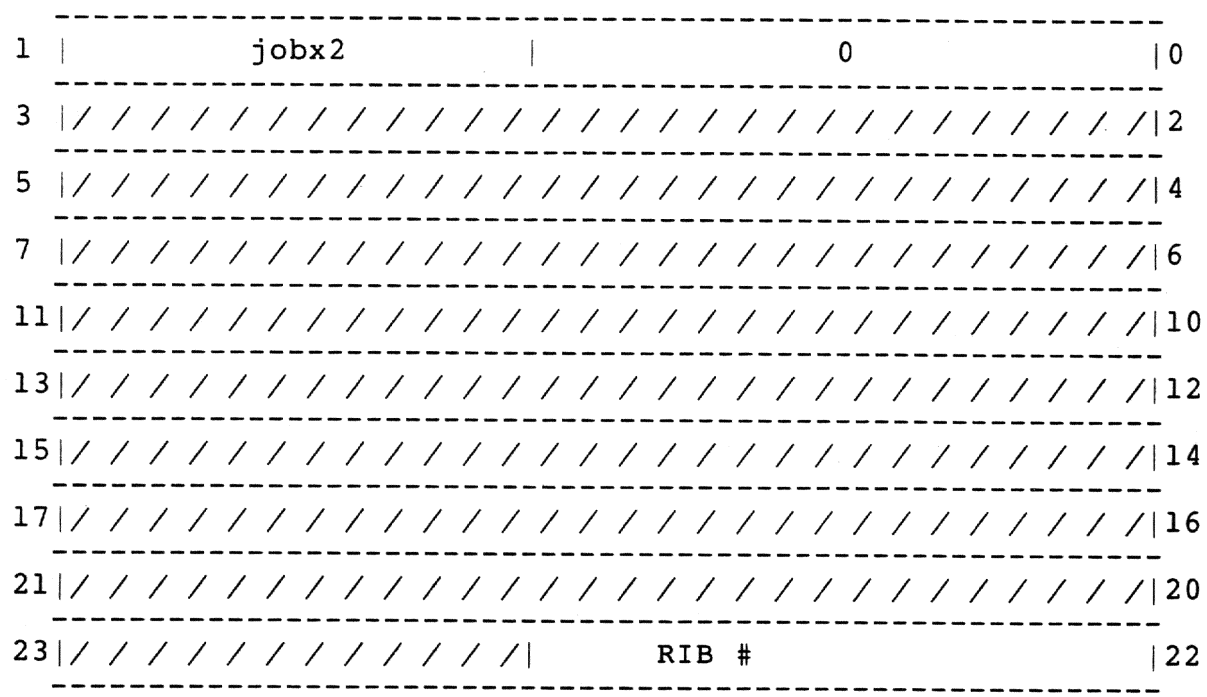
The label used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

> The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MREM call, the argument block contains:



jobx2

A zero byte for *jobx2* removes the calling job from the list of declared receivers. Privileged programs can remove another local RSTS/E job by specifying the job number times two.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
10	PRVIOL	The caller does not have JOBCTL privilege and has attempted to remove another job. That is, *jobx2* is nonzero.
18	BADFUO	The *jobx2* argument was odd. *Jobx2* must be zero to remove the calling job or job number times two to remove another job.

Example

The following example shows a typical Remove Receiver call:

```
PBLOCK:  .BLKW0   10.
            .
            .
            .
         .MREM    #PBLOCK,#0
```

The following example shows the $MREM form of the call:

```
PBLOCK:  $MREM    0
            .
            .
            .
         MOV      #PBLOCK,R0
         JSR      PC,$$$MSR
```

Get Local Node Parameters: NTLN (DECnet)

The NTLN call returns information to the calling program concerning the local node's network parameters. Data returned includes the node name and the default user's account to be used when an incoming Connect Initiate Message does not specify accounting information and the program has been defined as VERIFICATION PROGRAM. See the section "Automatic Job Start-Up" in Chapter 4 for a discussion of the default user's account.

MACRO Format

```
.NTLN area,bufptr,buflen,bufoff
```

or

```
label:  $NTLN bufptr,buflen,bufoff
```

Argument Descriptions

label

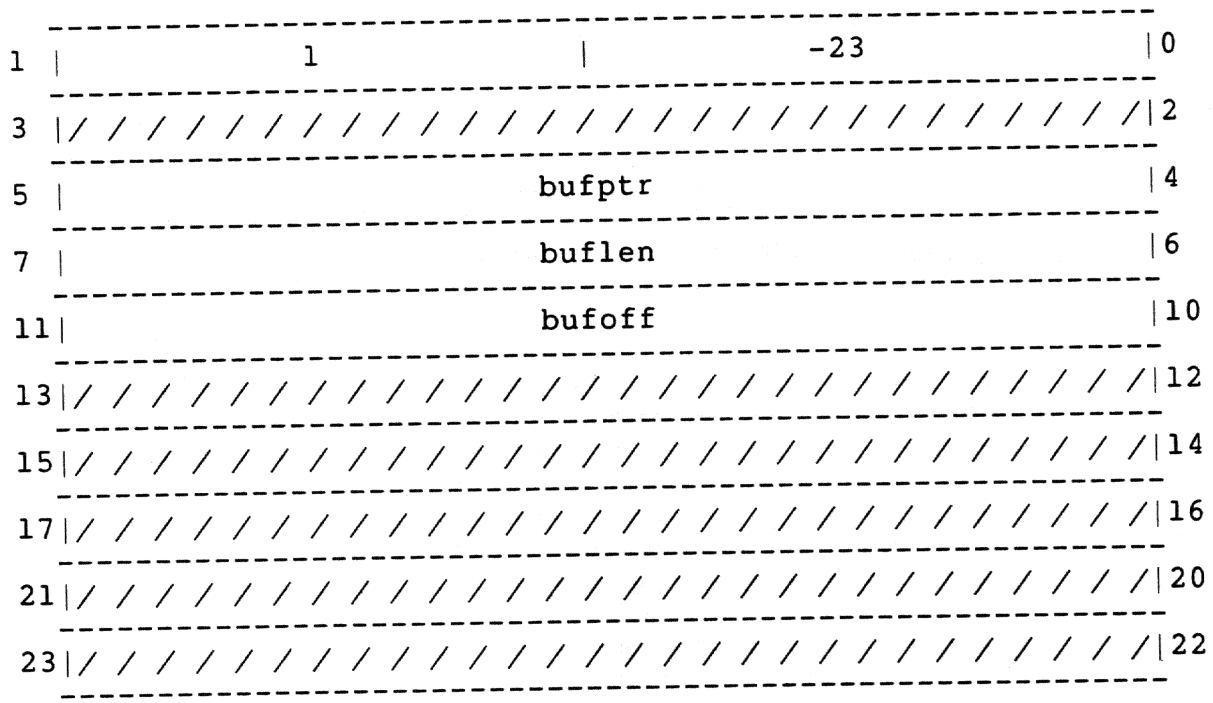
The label used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTLN call, the argument block contains:



bufptr

This word defines the starting address of the buffer to receive the local node information. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*. When the NTLN call completes successfully, data is returned to the buffer in the following format:

Bytes	Contents
0-5	Local node name, consisting of 1 to 6 uppercase, alphanumeric ASCII characters. The name contains at least one alphabetic character. A name shorter than 6 characters is left-justified and padded to 6 characters with spaces.
6-7	Reserved
8-13	Reserved
14-15	Local default project-programmer number. Byte 14 = programmer number; byte 15 = project number. If no default account is defined, both bytes are zero.

buflen

This word defines the number of bytes to be returned to the buffer specified with *bufptr*. If *buflen* is less than 16 (decimal) bytes, the call returns only as much data as there is room for. For example, if *buflen* = 8 bytes, the call returns only the local node name and address. Sixteen bytes are required to receive all the local node parameters.

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to *bufptr* to define the beginning address of the returned data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of 10. If the actual buffer, beginning at *bufptr* was only 16 bytes long, the returned data would overwrite 10 bytes beyond the end of the buffer.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
62	NORTS	DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a NTLN call used to obtain all the local node parameters. The node name is to be stored at location NAME; the address, at location ADDRSS; the node number, at location NUMBER; and the default project-programmer number, at location PPN. For example:

```
ARGBLK:   .BLKW0    10.             ;DEFINE ARGUMENT BLOCK

NAME:     .ASCII    /XXXXXX/        ;RESERVE SPACE FOR NODE NAME
ADDRSS:   .WORD     0               ;  NODE ADDRESS
NUMBER:   .ASCII    /XXXXXX/        ;  NODE NUMBER
PPN:      .BYTE     0,0             ;  PPN
            .
            .
            .
          .NTLN     #ARGBLK,#NAME,#16.,#0
```

The following example shows the $NTLN form of the same call:

```
ARGBLK:   $NTLN     NAME,16.,0      ;DEFINE ARGUMENT BLOCK

NAME:     .ASCII    /XXXXXX/        ;RESERVE SPACE FOR NODE NAME
ADDRSS:   .WORD     0               ;  NODE ADDRESS
NUMBER:   .ASCII    /XXXXXX/        ;  NODE NUMBER
PPN:      .BYTE     0,0             ;  PPN
            .
            .
            .
          MOV       #ARGBLK,R0
          JSR       PC,$$$MSR
```

Log User Event: NTEV (DECnet)

The NTEV call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system, in the standard Network Information and Control Exchange (NICE) protocol format. The system manager can use the normal network management SET and CLEAR LOGGING commands to control the filtering of these events. Optional data can accompany a user event.

Note

This system call performs a highly specialized function that requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture (DNA). It is provided as a convenience for the experienced network user and is not intended for normal network programs.

MACRO Format

```
.NTEV  area,evmod,evcls,evtyp,entyp,enval,bufptr,buflen,bufoff
```

or

```
label: $NTEV  evmod,evcls,evtyp,entyp,enval,bufptr,buflen,bufoff
```

Argument Descriptions

label

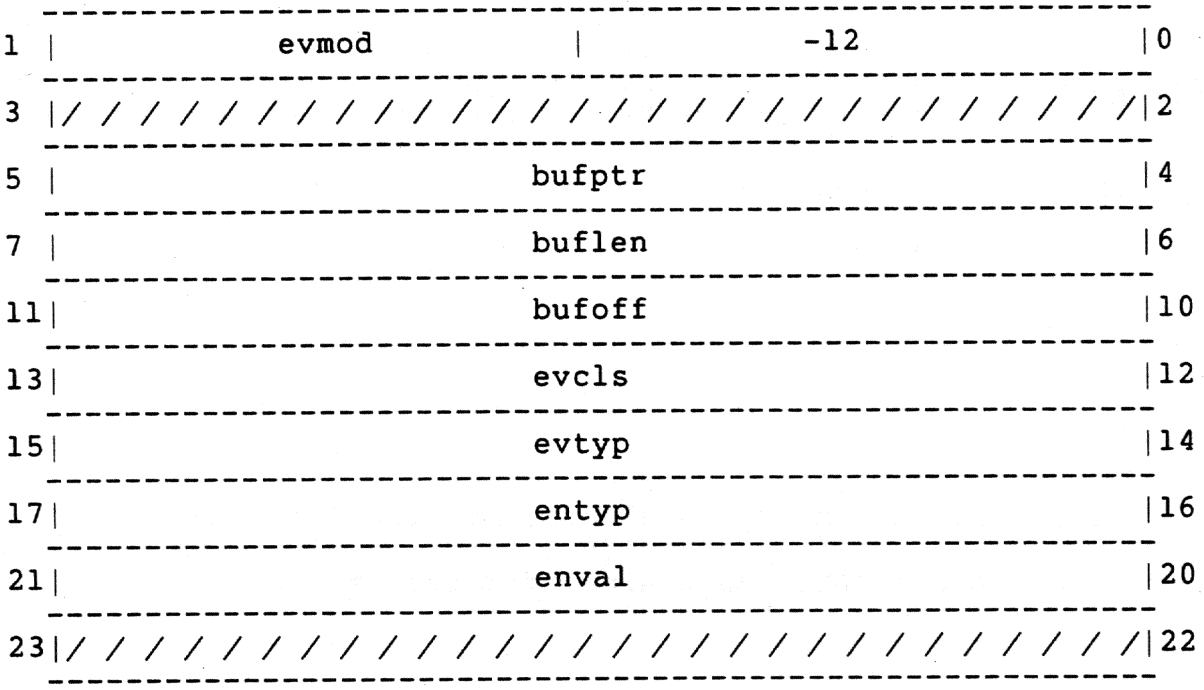
The label used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTEV call, the argument block contains:



evmod

This word parameter specifies what the system should do if it is unable to log the user event due to lack of resources. If *evmod* is set to zero, the call terminates with a status return of NOROOM (R0 = 4); the program can retry logging the event after a short delay. If *evmod* = 1, the system logs a Missed event message, and the call terminates with a successful status (R0 = 0).

Note that in the second case, the user data is not logged. The system merely logs the fact that it has lost some unidentified data. This is identical to the procedure followed with DECnet/E internal events when the event queue is full or there are no more small buffers.

evcls

This word parameter specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480 (decimal).

evtyp

This parameter specifies the type of event within the class specified by *evcls*. The event type itself is only a 5-bit value; thus, *evtyp* must be in the range of 0 to 31 (decimal).

entyp

This parameter specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E:

entyp	Meaning
-1	No specific entity
0	Node
1	Line
3	Circuit

enval

This parameter associates a value to the entity specified by the *entyp* parameter. If the entity is a node, set *enval* to the node address. If the entity is a line or a circuit, set *enval* to the address of the Device Data Block (DDB) maintained for the device.

bufptr

This word defines the starting address of the buffer containing the optional user data to be processed with the event. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*.

If you include this data, it must be in NICE protocol format. Basically, this means that the data should be formatted in a three-part field: a parameter type, a data type, and a parameter value. DECnet parameter types can be used if the event concerns a node, a circuit, or a line. In this case, the event logger outputs standard descriptive text for the specific parameter. If a DECnet parameter type is not used, parameter types in the range of 3900 to 4095 (decimal) can be specified. See the *DIGITAL Network Architecture Phase IV Network Management Functional Specification* for further information on event definitions and parameters.

Up to 200 (decimal) bytes of data can be passed to the event logger for processing.

buflen

This word defines the number of bytes of data in the buffer specified with *bufptr*. Up to 200 (decimal) bytes of data can be passed. If no data is specified, *buflen* must be 0.

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to *bufptr* to define the beginning address of the returned data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *bufptr*-100.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod* = 0.)
5	NOSUCH	The event logger is not running. Have the system manager start the event logger.
10	PRVIOL	The calling program must have SYSIO or SEND privilege at the time the NTEV call is issued.
18	BADFUO	The parameters passed are either inconsistent or invalid.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of a Local Link Address (*lla*) parameter number and value. A Missed event message is logged if the event itself cannot be.

```
ARGBLK:   .BLKW   10              ;DEFINE NTEV ARGUMENT BLOCK

MP$LLA    = 2130
ME$NOD    = 0
BUFOFF    = 6

EVBUF:    .BLKW   6
EVPARM:   .WORD   MP$LLA          ;STORAGE FOR EVENT DATA
EVLEN:    .BYTE   2
EVVAL:    .BLKB   2
          .
          .
          .
MOV       LLA,R0                  ;GET LLA NUMBER
MOVB      R0,EVVAL                ;STORE IN BUFFER
SWAB      R0
MOVB      R0,EVVAL+1
.NTEV     #ARGBLK,#1,#480.,#3,#ME$NOD,#135.,#EVBUF,#5,#BUFOFF
```

The following example shows the $NTEV form of the same call:

```
ARGBLK:   $NTEV   1,480.,3,ME$NOD,135.,EVBUF,5,BUFOFF

MP$LLA    = 2130
ME$NOD    = 0
BUFOFF    = 6

EVBUF:    .BLKW   6
EVPARM:   .WORD   MP$LLA          ;STORAGE FOR EVENT DATA
EVLEN:    .BYTE   2
EVVAL:    .BLKB   2
          .
          .
          .
MOV       LLA,R0                  ;GET LLA NUMBER
MOVB      R0,EVVAL                ;STORE IN BUFFER
SWAB      R0
MOVB      R0,EVVAL+1
MOV       #ARGBLK,R0              ;DO IT
JSR       PC,$$$MSR
```

The system calls in both the previous examples generate event messages in the following format:

```
Event type 480.3
Occurred 24-Sep-86 15:31:39.7 on node 3.135 (EDERE)
Node address 3.135 (EDERE)
Parameter #2130 = 40960
```

Send Calls

There are ten separate send calls -- two local and eight network. Each of these calls causes Session Control to format a message that is then either queued to a local program or transmitted across the network to a remote program.

Send Local Data Message: MSLD (Local)

The MSLD call sends up to 532 (decimal) bytes of user data to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, this program must have SEND privilege. The receiving program is identified in the MSLD call either by its job number or by the logical name with which it declared itself a receiver.

MACRO Format

.MSLD *area,job,namptr,bufptr,buflen,bufoff,parptr,parlen*

or

label: $MSLD *job,namptr,bufptr,buflen,bufoff,parptr,parlen*

Argument Descriptions

label

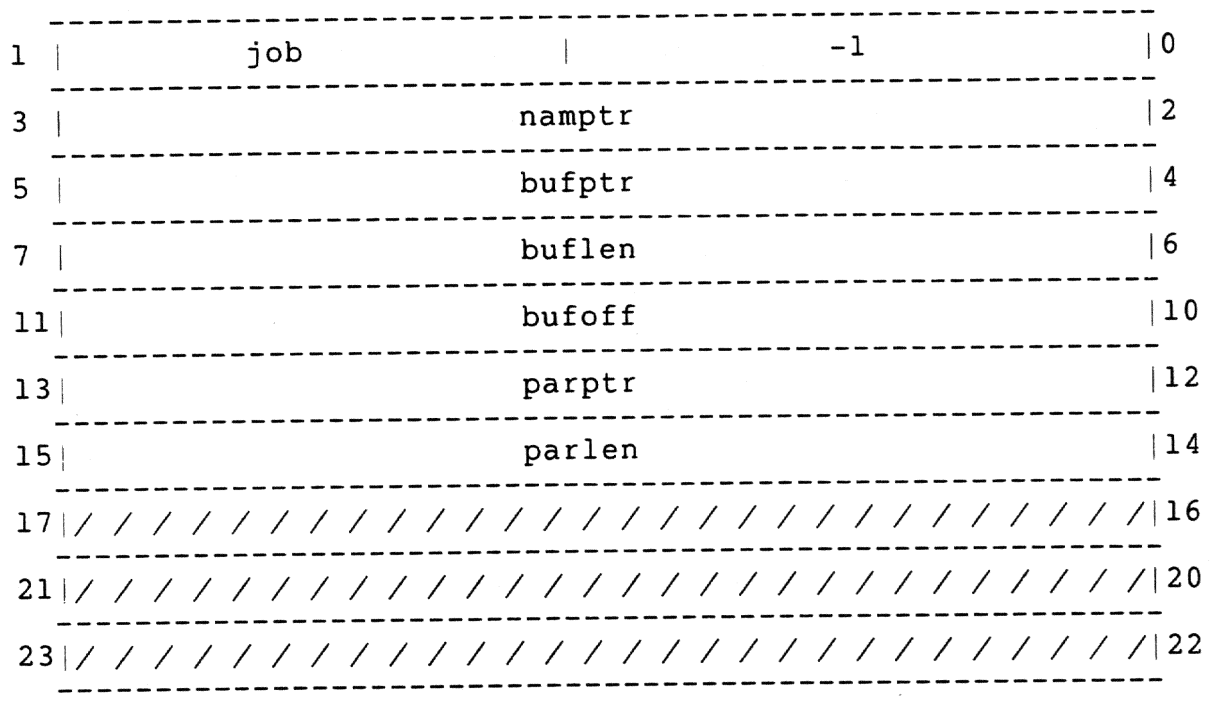
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

> The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MSLD call, the argument block contains:



job

If *job*=0, the call uses the logical name to determine the receiver. If *job*=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*=8 directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data

passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

namptr

This word is the address of a 6-byte area containing the logical name of the local program to which the data is to be sent. The logical name itself consists of one to six ASCII characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). If the name is less than six characters, it must be left-justified in the 6-byte area and the remaining bytes must be filled with spaces. Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver

name, the send will succeed only if the name is associated with the job number.

bufptr

This word defines the starting address of the message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*.

buflen

This word defines the number of bytes to be sent from the *bufptr* buffer. Possible values for *buflen* range from 0. through 512.

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the message data to be sent.

Note

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512. and a *bufoff* of 1000. The MSLD call would send 512. bytes beginning at *bufptr*+1000.

parptr

This word is the starting address of an area containing up to 20. bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the *bufptr* area. For example, a MACRO program receiving a Local Data Message gets the parameter message data in the FIRQB and the other data in a buffer defined in the Receive call.

parlen

This word defines the number of bytes to be sent from the parameter area whose starting address is defined by *parptr*. The *parlen* value can range from 0. through 20.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	NOSUCH	The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a Declare Receiver call) before any data can be transmitted to it.
10	PRVIOL	Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	BADFUO	The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	BADCNT	Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0. through 512. The *parlen* argument can range from 0. through 20.
32	NOBUFS	System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Example

The following example shows a local send to job number 23. A buffer length of 6. and offset of 18. indicate that the characters OFFSET are sent.

```
ARGBLK:  .BLKW0   10.
BUFFER:  .ASCII   /TO ILLUSTRATE THE OFFSET/
           .
           .
           .
.msld    #argblk,#46.,,#buffer,#6.,#18.,#0,#0
```

The following example shows the $MSLD form of the same call:

```
ARGBLK:  $MSLD    46.,,BUFFER,6.,18.
           .
           .
           .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Send Local Data Message with Privilege Mask: MSPR (Local)

The MSPR call sends up to 524 (decimal) bytes of user data, plus a privilege mask supplied by the monitor, to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, this program must have SEND privilege. The receiving program is identified in the MSPR call either by its job number or by the logical name with which it declared itself a receiver.

MACRO Format

```
.MSPR area,job,namptr,bufptr,buflen,bufoff,parptr,parlen
```

or

```
label: $MSPR job,namptr,bufptr,buflen,bufoff,parptr,parlen
```

Argument Descriptions

label

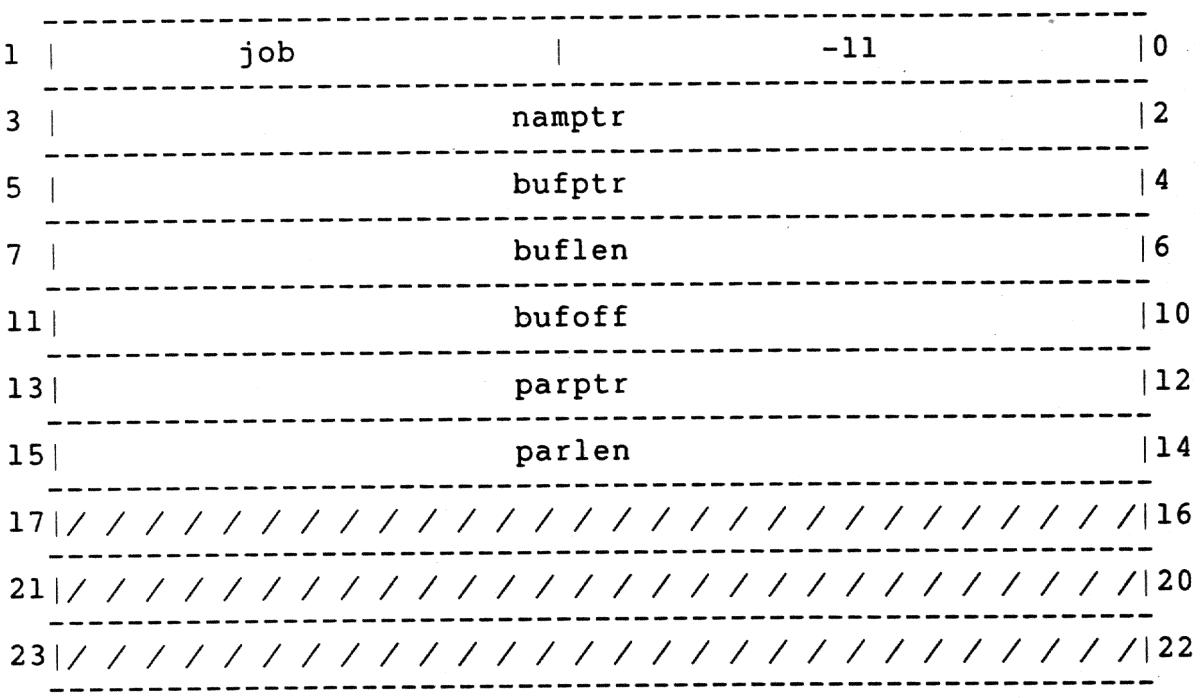
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

> The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MSPR call, the argument block contains:



job

If *job*=0, the call uses the logical name to determine the receiver. If *job*=128+LOT, the call uses the local object type (LOT) specified. Only single-instance object types are valid. Otherwise, *job* can be set to the job number times 2 of the local receiving job. For example, specifying *job*=8 directs a message to job 4.

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If *job* is nonzero and even, and the name field is null, the call interprets it as the job number (times 2) of the intended receiver.

If the name field is not null, the call attempts to send the message to the receiver whose logical name matches the name field of the data

passed. If *job* is zero, the message is sent to the named receiver. If *job* is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in *job*. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If *job* is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a RIB value of zero.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

namptr

This word is the address of a 6-byte area containing the logical name of the local program to which the data is to be sent. The logical name itself consists of one to six ASCII characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underscore). If the name is less than six characters, it must be left-justified in the 6-byte area and the remaining bytes must be filled with spaces. Leading or embedded spaces are invalid. If *job* is nonzero and you specify a receiver

name, the send will succeed only if the name is associated with the job number.

bufptr

This word defines the starting address of the message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*.

buflen

This word defines the number of bytes to be sent from the *bufptr* buffer. Possible values for *buflen* range from 0. through 512.

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the message data to be sent.

Note

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512. and a *bufoff* of 1000. The MSPR call would send 512. bytes beginning at *bufptr*+1000.

parptr

This word is the starting address of an area containing up to 12. bytes of parameter message data to be sent to the local program. This data is delivered separate from the message data in the *bufptr* area. For example, a MACRO program receiving a Local Data Message gets the parameter message data in the FIRQB and the other data in a buffer defined in the Receive call.

parlen

This word defines the number of bytes to be sent from the parameter area whose starting address is defined by *parptr*. The *parlen* value can range from 0. through 12..

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	NOSUCH	The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a Declare Receiver call) before any data can be transmitted to it.
10	PRVIOL	Some access violation has occurred. Either the receiver does not allow any local senders or the sender does not have SEND privilege and the receiver allows only privileged senders.
18	BADFUO	The value in *job* is odd. It must be equal to zero or to the receiver's job number times two.
31	BADCNT	Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512. The *parlen* argument can range from 0 through 12.
32	NOBUFS	System buffers are currently not available to store this message for the intended local receiver. A retry may succeed.

Send Connect Initiate Message: NTCI (DECnet)

The NTCI call requests a logical link to another program. The call establishes the user link address that the calling program uses to refer to the link. The other program is identified with a Connect Data Block. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16. bytes of user data can be sent with the NTCI call.

MACRO Format

.NTCI *area,ula,llmod,rmax,bufptr,buflen,bufoff,rib*

or

label: $NTCI *ula,llmod,rmax,bufptr,buflen,bufoff,rib*

Argument Descriptions

label

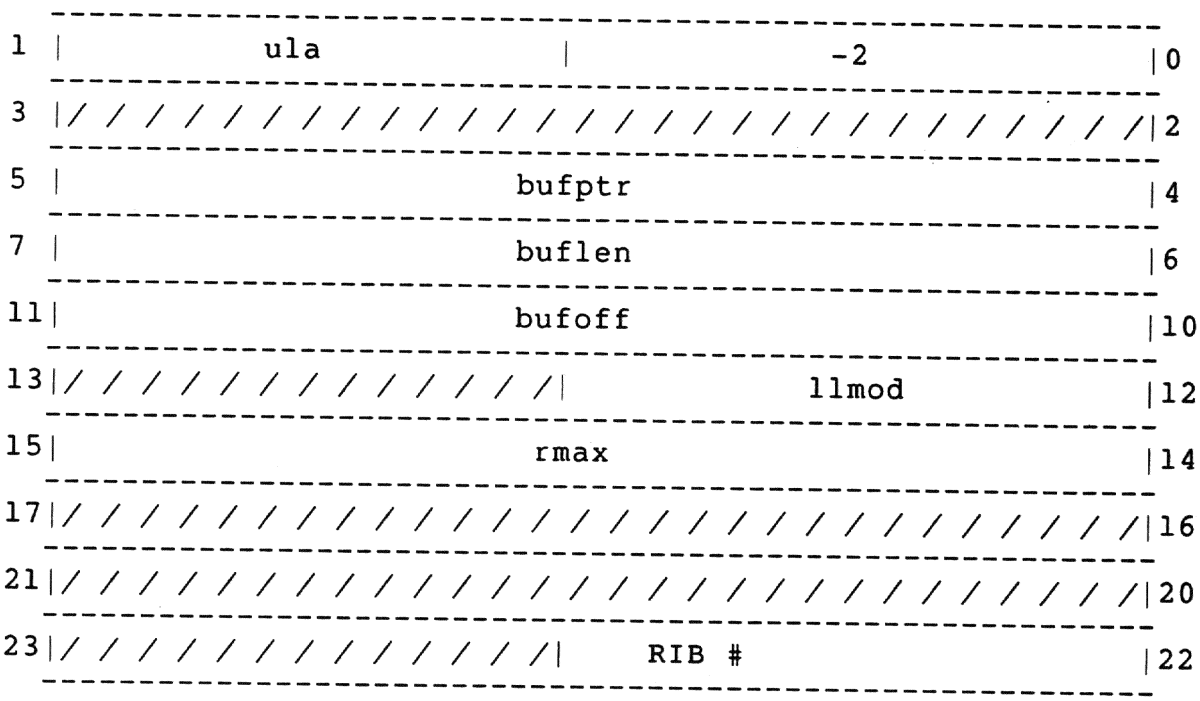
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCI call, the argument block contains:



ula

The User Link Address (*ula*) is a byte value in the range of 1. to 255. (1 to 377 octal). Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI is accepted by the remote ECL and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

llmod

The value of the *llmod* byte indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

llmod	Meaning
0	No flow control
1	Segment flow control
2	Message flow control

Chapter 4 describes flow control in detail. Remember that both programs on a logical link select their own flow control independent of each other. The *llmod* value here is the selection made by the program issuing the call. If *llmod* is nonzero, the calling program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The word value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default. The calling program is informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive (MRCV)" for more information.

bufptr

This word defines the starting address of the Connect Data Block and optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

The Connect Data Block identifies the target program for the requested logical link. Figure 11-1 shows the format of the Connect Data Block. Table 11-3 describes the fields within the block. Up to 16 (decimal) bytes of user data can follow the Connect Data Block.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0

through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

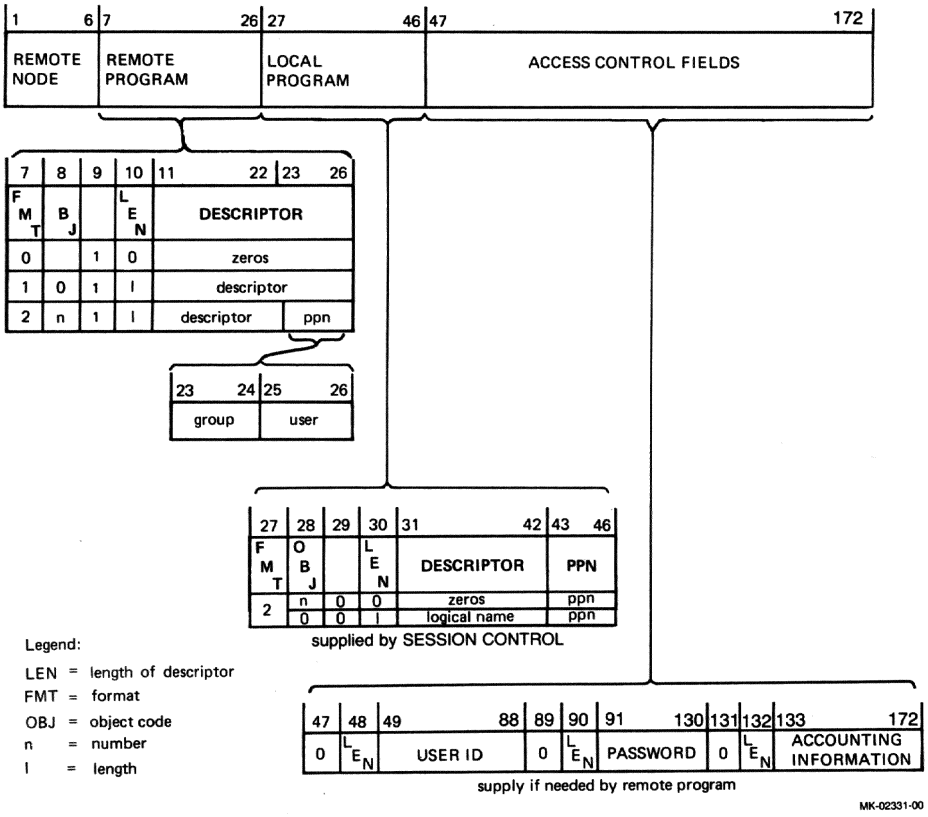


Figure 11-1: Format of Connect Data Block on Send Connect Initiate Message

Table 11-3: Format of Connect Data Block on Send Connect Initiate Message

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node to which the Connect Initiate Message is to be sent. Names shorter than 6 characters must be left-justified with space fill to 6 characters. Leading and embedded spaces are invalid. If no node name is specified (that is, the request is directed to a program at the local node), these 6 bytes contain an ASCII 0 and 5 spaces.
7-26	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of three formats: **Format 0** The remote program is identified by object type code alone. The fields are:

Bytes	Content
7	Zero, to indicate a format 0 name.
8	Object type code of the remote program.
9	Must be one.
10-26	Zero (Not used).

Table 11-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	Format 1 The remote program is identified by a descriptor. The fields are: Bytes — Content 7 — One, to indicate a format 1 name. 8 — Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). 9 — Must be one. 10 — Length of descriptor, 0 to 16 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field. 11-26 — Remote program descriptor (left-justified). For a DECnet/E program, this is the logical name that the remote program defined in its Declare Receiver call. For other systems, it is the descriptor used to register the program for network operations. Format 2 This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program for which a connect request is intended. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

Table 11-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The subfields are:
	Bytes — **Content**
	7 — Two, to indicate a format 2 name.
	8 — Object type code of the remote program, if applicable. If zero, the descriptor must be used.
	9 — Must be one.
	10 — Length of descriptor, 0 to 12 bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
	11-22 — Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
	23-24 — Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E PPN.
	25-26 — Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E PPN.
27-46	**Local Program.** This field is filled in by the local session control using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

Table 11-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
	The information is passed to the remote system in format 2, as described previously. Byte 27 is filled in with a value of 2. Byte 29 is always zero. If the program declared its identity with an object code alone, the object code is passed in byte 28 and bytes 30-42 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in bytes 31-42 and byte 28 is zero. The length of the logical name is given in byte 30. The RSTS/E PPN under which the program is running is passed in bytes 43-46 as the group and user codes: the project number is in byte 43; the programmer number is in byte 45. Both are in binary format.
47-172	**Access Control Fields.** These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number or system password, so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved -- must be zero.
48	Length of user identification information specified in the next field, 0 to 40 bytes.

Table 11-3: Format of Connect Data Block on Send Connect Initiate Message (Cont.)

Bytes	Content
49-88	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, set the length (byte 48) to zero.
89	Reserved -- must be zero.
90	Length of password information specified in the next field, 0 to 40 bytes.
91-130	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login. A password supplied here must be acceptable to the remote system or program, if one is required. If not required, set the length (byte 90) to zero.
131	Reserved -- must be zero.
132	Length of the accounting information specified in the following field, 0 to 40 bytes.
133-172	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.

buflen

This word defines the number of bytes to be sent. Since the Connect Data Block is 172. bytes long and the optional user data can be from 0. to 16. bytes long, acceptable values for *buflen* range from 172. through 188.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the Connect Data Block.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 172. and a *bufoff* of -100. The beginning address of the Connect Data Block would then be interpreted as *bufptr*-100.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
3	INUSE	The calling program already has a logical link with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.
6	NODEVC	The node named in the Connect Data Block is not known to the system.
14	HNGDEV	The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	DTOOOF	ECL cannot allocate its Local Link Address (*lla*) for the logical link. The maximum number of logical links allowed at the local RSTS/E node (63 or some limit between 1 and 63 -- set by the system manager) has been reached. See the system manager if the limit in effect is inadequate. A retry may succeed.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
18	BADFUO	One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the calling program to the remote system. Issue a Remove Receiver call, followed by another Declare Receiver call. o An invalid value was detected in one of the following fields: - Remote program field in the Connect Data Block. - One of the three access control fields in the Connect Data Block. - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255 (1 to 377 octal). - The local link modifier (*llmod*) value must be 0, 1, or 2. - The remote descriptor length (octal byte offset 11 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).
22	PAKLCK	Session Control cannot create a new logical link because the physical link to the remote node or the local node itself has been scheduled for shutdown by the system manager.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 120 and 136 (decimal) bytes for a Connect Initiate Message.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
62	NORTS	DECnet/E has not been enabled. Normal DECnet cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

This example shows a NTCI call used to direct a Connect Initiate Message to a program with object type code 140 at node MIAMI. The Connect Data Block uses a format 0 name to address the remote program. The local program name is supplied by ECL and no access control fields are being used. Thus, the rest of the Connect Data Block is filled with zeros to make up 172. bytes. The user link address is 15., segment flow control is requested, and the receive maximum is 512. bytes. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0   10.              ;DEFINE ARGUMENT BLOCK
CDB:     .ASCII   /MIAMI /
         .BYTE    0,140.
         .BLKW0   114.
            .
            .
            .
         .NTCI    #ARGBLK,#15.,#0,#512.,#CDB,#172.,#0,#5
```

The following example shows the $NTCI form of the same call:

```
ARGBLK:  $NTCI    15.,0,512.,CDB,172.,0,5
CDB:     .ASCII   /MIAMI /
         .BYTE    0,140.
         .BLKW0   114.
            .
            .
            .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Send Connect Confirm Message: NTCC (DECnet)

The NTCC call accepts a remote program's request for a logical link. The call establishes the User Link Address (ULA) that the calling program uses to refer to the link. The Local Link Address (LLA) assigned by the local ECL identifies the particular link being accepted. The desired flow control option and the receive maximum for the link are specified, and up to 16. bytes of message data can be passed to the remote program.

MACRO Format

.NTCC *area,ula,lla,llmod,rmax,bufptr,buflen,bufoff,rib*

or

label: $NTCC *ula,lla,llmod,rmax,bufptr,buflen,bufoff,rib*

Argument Descriptions

label

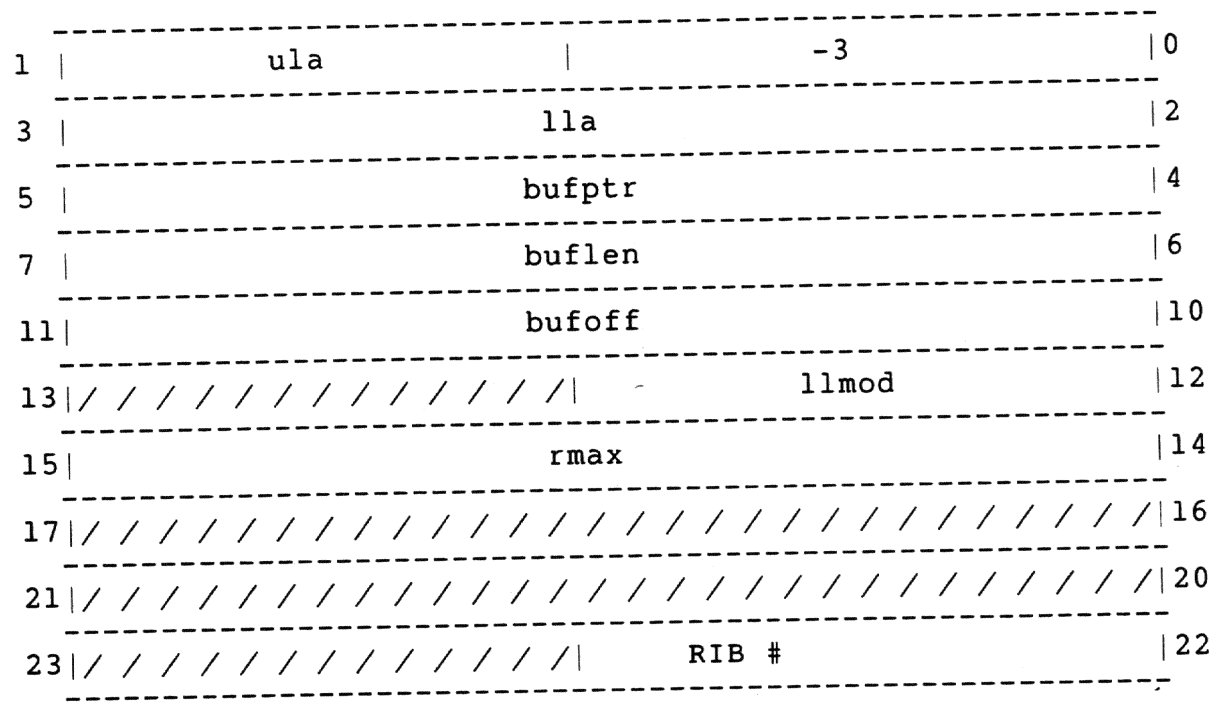
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCC call, the argument block contains:



ula

The User Link Address (*ula*) is a byte value in the range of 1. to 255. (1 to 377 octal). Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The Local Link Address (*lla*) is a word value that identifies the link being accepted. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in the FIRQB at octal offset 6 when the program receives a Connect Initiate Message. The section "Format of Received Connect Initiate Message" describes the information returned for a received Connect Initiate Message.

llmod

The value of the *llmod* byte indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are:

llmod	Meaning
0	No flow control
1	Segment flow control
2	Message flow control

Chapter 4 describes flow control in detail. Remember that both programs in a logical link select their own flow control independent of each other. The *llmod* value here is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The word value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by ECL itself limits the amount of data that can be passed in a single Network Data Message. If the maximum size set for ECL by Network Management is not large enough to handle the receive maximum as specified in *rmax*, the local ECL alters the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, ECL supplies the size of its receive buffers as a default value.

The remote ECL uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument in the section "Receive (MRCV)" for a description of how to process large messages in small pieces.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0. to 16.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16. and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
3	INUSE	The calling program already has a logical link with the same User Link Address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. (The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established).
10	PRVIOL	Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Confirm Message for the same logical link causes this error because the link is already established.
18	BADFUO	One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o An invalid value was detected in one of the following fields: - The User Link Address (*ula*) given was zero. Valid values range from 1 to 255 (1 to 377 octal). - The local link modifier (*llmod*) value must be 0, 1, or 2.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 (decimal) bytes for a Connect Confirm Message.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, a NTCC call responds to a received Connect Initiate Message. No intervening calls to the monitor have been made that would have destroyed the *lla* in the FIRQB so this argument is specified as FIRQB+6. The call also establishes a ULA of 15., a receive maximum of 256. bytes, and indicates that the calling program requests message flow control. No user message data is sent so, to make sure that no other arguments remain in the argument block from a previous call, these arguments are zero. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0   10.              ;DEFINE ARGUMENT BLOCK
            .
            .
            .
         .NTCC    #ARGBLK,#15.,@#FIRQB+6,256.,#0,#0,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTCC   15.,,256.,,,5
           .
           .
           .
         MOV     #ARGBLK,R0
         MOV     @#FIRQB+6,2(R0)
         JSR     PC,$$$MSR
```

Note that with the "$" form, the *lla* value must be moved from FIRQB+6 to the appropriate place in the argument block at execution time. The *lla* value is not known at assembly time when the argument block is generated by the "$" form.

Send Connect Reject Message: NTCR (DECnet)

The NTCR call rejects a remote program's request for a logical link. The link is identified by the Local Link Address (LLA) established by the local ECL. Up to 16 (decimal) bytes of user message data can be sent to the remote program as part of the Connect Reject Message.

MACRO Format

.NTCR *area,lla,reason,bufptr,buflen,bufoff,rib*

or

label: $NTCR *lla,reason,bufptr,buflen,bufoff,rib*

Argument Descriptions

label

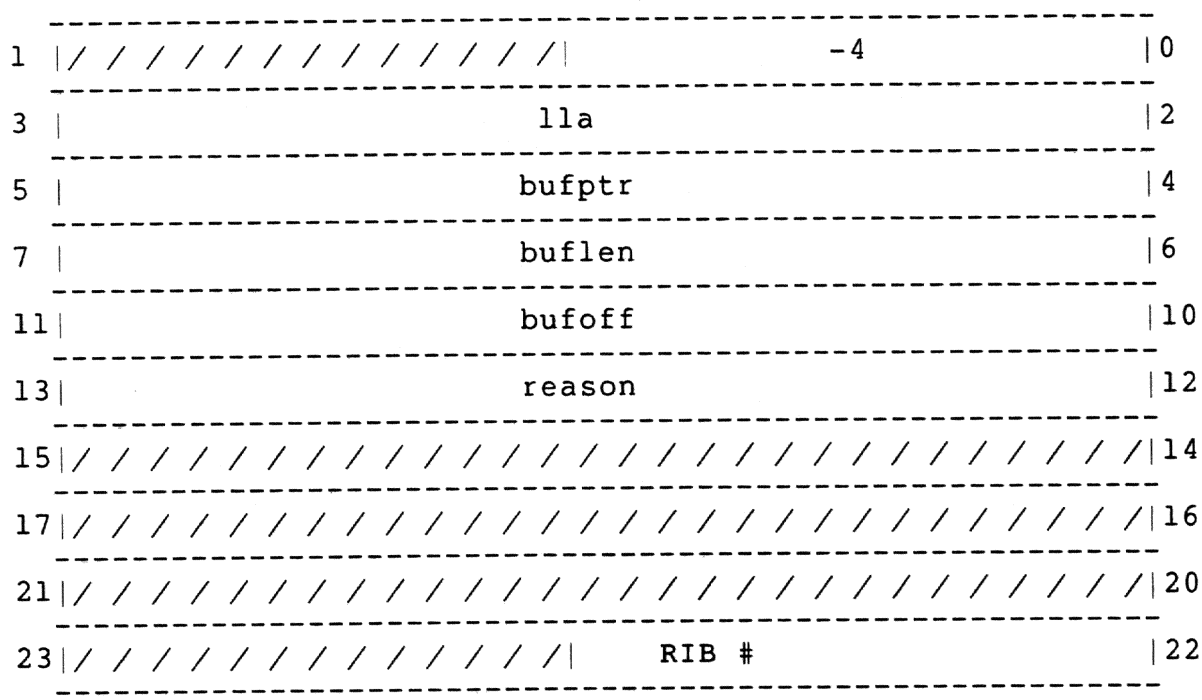
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCR call, the argument block contains:



lla

The Local Link Address (*lla*) is a word value that identifies the link being rejected. It is a number assigned by the local ECL when it received the Connect Initiate Message requesting the link. The LLA is passed to the local program in the FIRQB at octal offset 6 when the program receives a Connect Initiate Message. The section "Format of Received Connect Initiate Message" describes the information returned for a received Connect Initiate Message.

reason

DECnet/E NSP allows one of three reason code values for this word: 0, 34 (decimal), or 36 (decimal).

Most user programs use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent from the buffer. Possible values of *buflen* range from 0. to 16.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16. and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The Local Link Address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the last case, a Link Abort Message has already been queued for this link. (The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established).
10	PRVIOL	Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation causes this error. For example, a second Connect Confirm Message for the same logical link causes this error because the link is already established.
18	BADFUO	One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The reason code supplied with the Connect Reject Message was not 0, 34 (decimal), or 36 (decimal).
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 (decimal) bytes for a Connect Confirm Message.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

In this example, a NTCR call responds to a received Connect Initiate Message. No intervening calls to the monitor have been made that would have destroyed the *lla* in the FIRQB so this argument is specified as FIRQB+6. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0   10.              ;DEFINE ARGUMENT BLOCK
REASON:  .ASCII   /TOO LATE/       ;DEFINE MESSAGE DATA
           .
           .
           .
         .NTCR    #ARGBLK,@#FIRQB+6,#0,#REASON,#8.,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTCR    ,,,REASON,8.,,5
REASON:  .ASCII   /TOO LATE/
           .
           .
           .
         MOV      #ARGBLK,R0
         MOV      @#FIRQB+6,2(R0)
         JSR      PC,$$$MSR
```

Note that with the "$" form, the *lla* value must be moved from FIRQB+6 to the appropriate place in the argument block at execution time. The *lla* is not known at assembly time when the "$" form generates the values for the argument block.

Send Network Data Message: NTDM (DECnet)

The NTDM call transmits user data over an established logical link. The amount of user data you can transmit in one Network Data Message is called a segment. You can set a flag byte to indicate whether the segment is the end of a logical message. If the remote program requested flow control when the link was established, requests for data must be outstanding or this call results in an error.

MACRO Format

.NTDM *area,ula,dmflgs,bufptr,buflen,bufoff,rib*

or

label: $NTDM *ula,dmflgs,bufptr,buflen,bufoff,rib*

Argument Descriptions

label

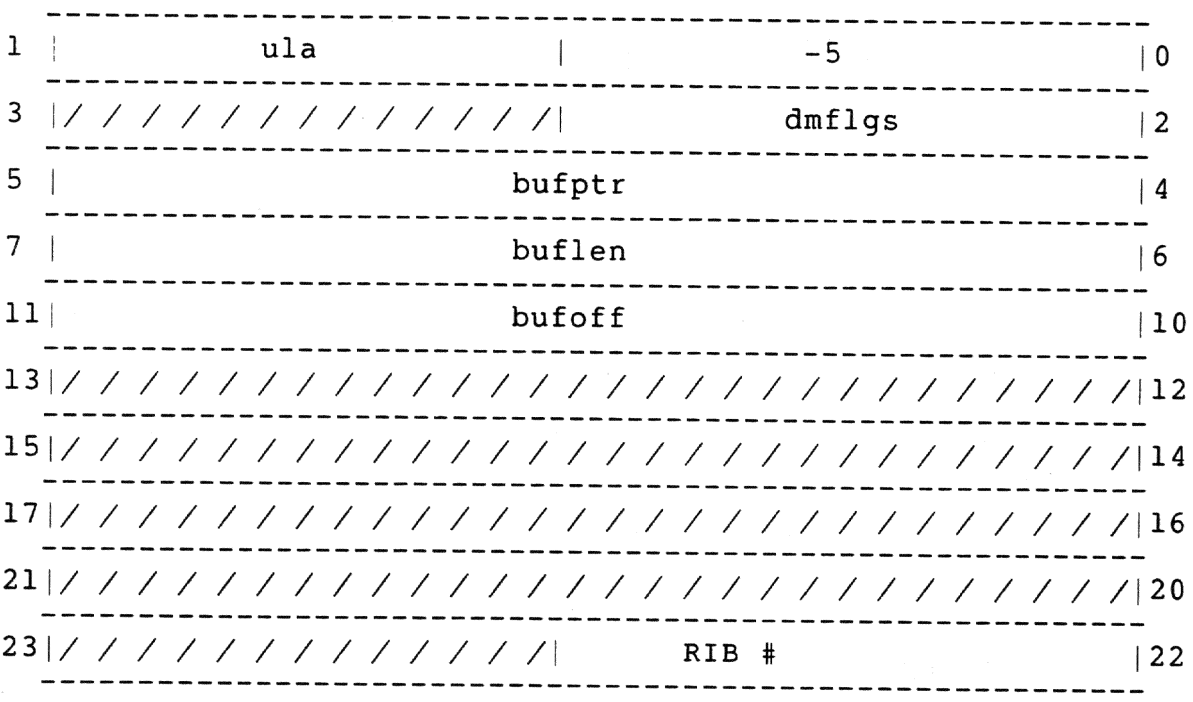
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTDM call, the argument block contains:



ula

The *ula* is a byte value specifying the User Link Address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The low-order two bits of the *dmflgs* byte indicate whether the segment being sent is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, ECL uses these two bits to regulate the transmission of logical messages as the remote program requests them. Chapter 4 for a full discussion of flow control. The possible outcomes are:

Bit 1	Bit 0	Meaning
0	0	Middle segment of a logical message
0	1	First segment of a logical message
1	0	Last segment of a logical message
1	1	Sole segment of a logical message

DECnet/E ECL uses bit 1 to keep its count of logical messages transmitted to the remote program. Bit 0 is not used in maintaining the logical message request count but is passed to the remote system and program.

ECL also uses bit 1 to determine whether it permits a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have bit 1 set (=1), ECL assumes that the last message has not been completely sent and does not permit the disconnect. This is always done, regardless of the flow control option selected by the receiver.

Some DECnet implementations require that bit 1 be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, bit 1 should always be set. Setting bit 1 in this case does not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

When the sign bit (bit 7) of the *dmflgs* byte is set to 1, link status information is returned to the FIRQB when the NTDM call completes successfully. If link status information is requested, it is returned in the same format as a received Link Service Message (see the section "Format of Received Link Service Message").

A useful way to document and set these flags is:

```
EOM =   2.
BOM =   1.
RLS = 128.
```

Then, to indicate the end of a logical message and to request the return of link status information, the *dmflgs* argument in the call could be specified as #EOM+RLS.

bufptr

This word defines the starting address of the user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes to be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or the remote ECL. It is passed on to the calling program as a transmit maximum (the *tmax* value in a received Connect Initiate or Connect Confirm Message). Thus, the value of *buflen* can range from 1 to *tmax*.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100. and a *bufoff* of 20. One hundred bytes beginning at *bufptr*+20. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The Data Message transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The User Link Address (*ula*) does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Network Data Message Message over a logical link that has not yet been confirmed causes this error.
18	BADFUO	One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
19	INTLCK	Either the remote system has inhibited transmission on this link because of a backpressure condition, or there are no outstanding requests for message data from the remote program (assuming it requested segment or message flow control for the link). The calling program is notified with a Link Service Message when the condition clears, as described in Chapter 4.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows a 100-byte send from a buffer area. The ULA is 20. The value of 203 for *dmflgs* identifies the message data as the sole segment of a logical message and requests the return of link status information. RIB #5 is used. For example:

```
ARGBLK:   .BLKW0    10.      ;DEFINE ARGUMENT BLOCK
BUFFER:   .BLKW0    100.     ;DEFINE BUFFER FOR SEND
            .
            .
            .
(fill buffer with data to be sent)
            .
            .
            .
          .NTDM    #ARGBLK,#20.,#203,#BUFFER,#100.,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:   $NTDM     20.,203,BUFFER,100.,,5
BUFFER:   .BLKW0    100.
            .
            .
            .
(fill buffer with data to be sent)
            .
            .
            .
          MOV       #ARGBLK, R0
          JSR       PC,$$$MSR
```

Send Interrupt Message: NTIN (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message is placed at the head of the pending message queue, behind the first message (since it may already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt messages are subject to flow control (see Chapter 4).

MACRO Format

```
.NTIN area,ula,inflgs,bufptr,buflen,bufoff,rib
```

or

```
label:  $NTIN ula,inflgs,bufptr,buflen,bufoff,rib
```

Argument Descriptions

label

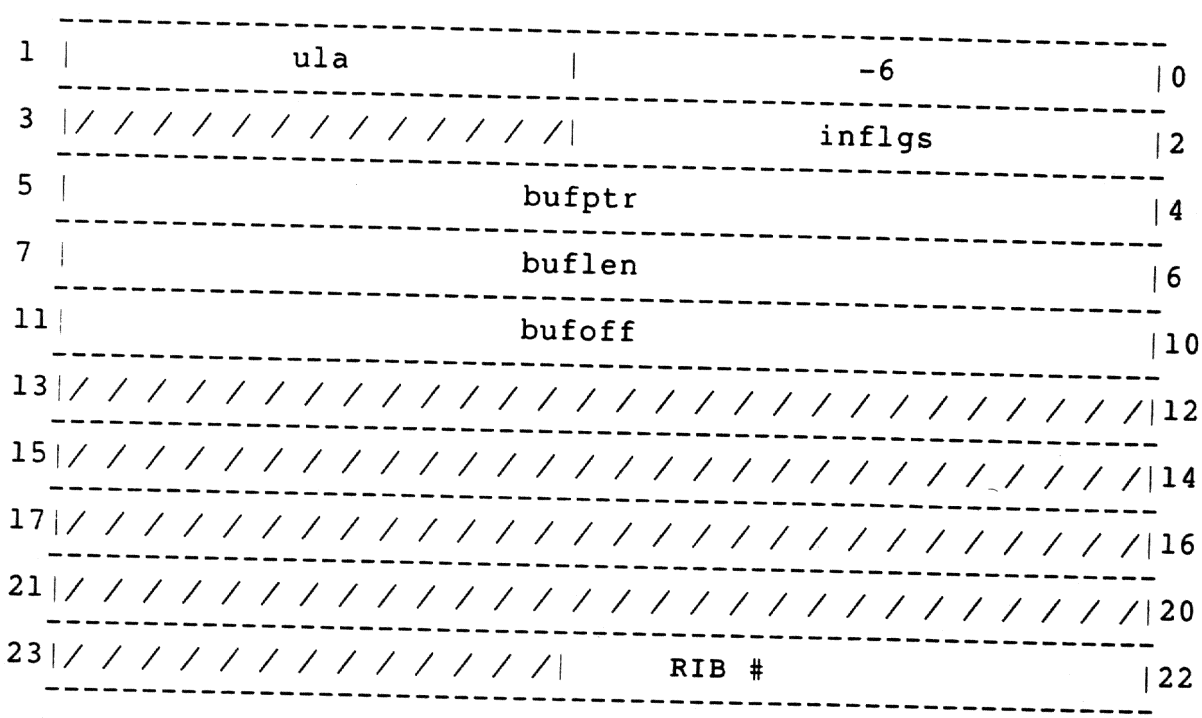
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTIN call, the argument block contains:



ula

The *ula* is a byte value specifying the User Link Address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

If the sign bit (bit 7) of the *inflgs* byte is set to 1, link status information is returned to the FIRQB when the NTIN call completes successfully. If link status information is requested, it is returned in the same format as a received Link Service Message (see the section "Format of Received Link Service Message").

A useful way to document and set this flag is to assign a mnemonic to the value 128 (decimal) to set or clear the flag, as follows:

```
RLS = 128
```

Then, to request the return of link status information, the *inflgs* argument can be specified as #RLS.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0. to 16.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address user message data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16. and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	NOSUCH	The User Link Address (*ula*) does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed causes this error.
18	BADFUO	One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The User Link Address *ula* given was zero. Valid values range from 1 to 255 (1 to 377 octal).
19	INTLCK	The remote ECL has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message is delivered to the calling program when the condition clears. See Chapter 4 for a full discussion of flow control.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 (decimal) bytes for an Interrupt Message.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example shows an NTIN call. The user data "TIMESUP" is passed to a remote program over logical link 15. Note that the remote program would have to be coded to recognize this message and take appropriate action. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0   10.            ;DEFINE DATA BLOCK
BUF:     .ASCII   /TIMESUP/      ;DEFINE USER DATA
           .
           .
           .
         .NTIN    #ARGBLK,#15.,#0,#BUF,#7.,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTIN    15.,0,BUF,7.,,5
BUF:     .ASCII   /TIMESUP/
           .
           .
           .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Send Link Service Message: NTLS (DECnet)

The NTLS call:

- Requests data from a remote program over a flow-controlled logical link (see Chapter 4)
- Reenables incoming Interrupt Messages over a logical link (see Chapter 4)
- Obtains status information for the link

No user data is allowed with a Link Service Message.

MACRO Format

.NTLS *area,ula,lsflgs,drcnt,ircnt,rib*

or

label: $NTLS *ula,lsflgs,drcnt,ircnt,rib*

Argument Descriptions

label

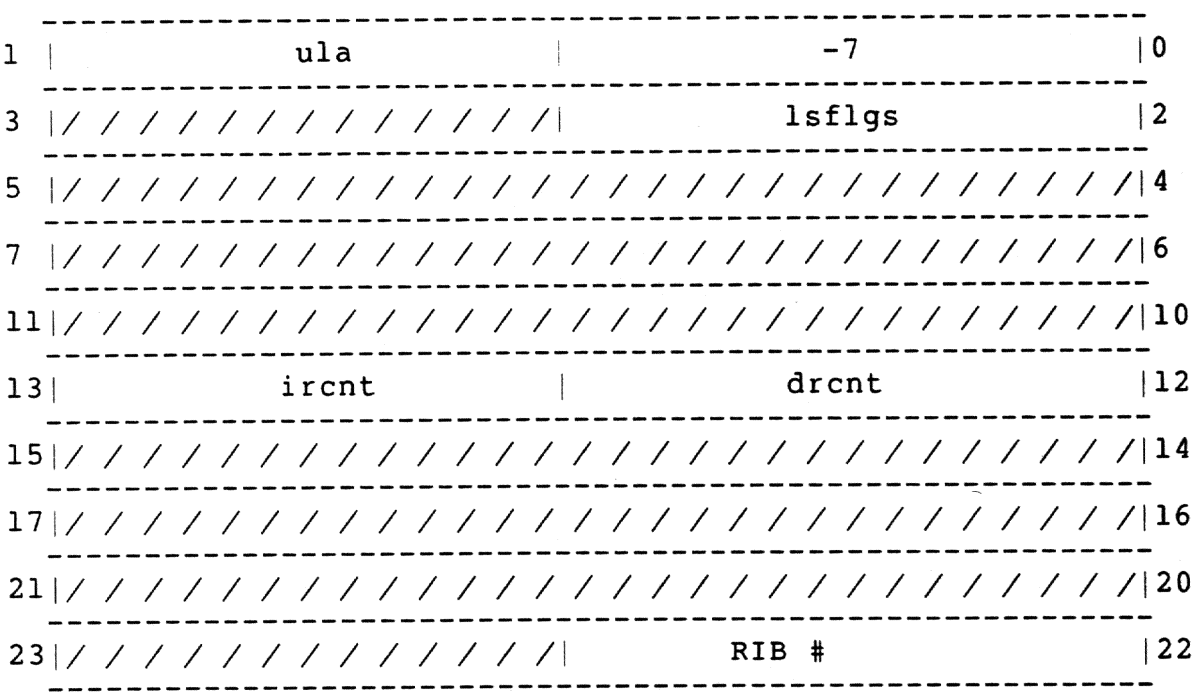
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTLS call, the argument block contains:



ula

The *ula* is a byte value specifying the User Link Address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The low-order two bits of this byte indicate the purpose of the call.

Bit 1	Bit 0	Meaning
0	0	Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested message or segment flow control in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message for this logical link. If the program requested no flow control, an NTLS call with the low-order two bits of *lsflgs* = 00 results in an error (BADFUO).

0	1	Indicates the call is to reenable incoming Interrupt Messages for this link. In this case the *ircnt* argument is interpreted as the number of interrupts being requested (always 1).
1	0	Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

The sign bit of this byte is significant in the first two previous cases. When the sign bit (bit 7) is set to 1, link status information is returned to the FIRQB when the NTLS completes successfully.

Link status information is always returned when bit 1 is set. If it is requested, the information is returned to the FIRQB in the same format as a received Link Service Message (see the section "Format of Received Link Service").

A useful way to document and set these flags is to assign mnemonics to values that set (or clear) the appropriate bits. For example:

```
REQMSG =   0.
REQINT =   1.
RLS    = 128.
```

Now you can request a data message and return link status information by specifying the *lsflgs* argument as #REQMSG+RLS.

drcnt

The data request count byte (*drcnt*) is meaningful only when bits 0 and 1 of *lsflgs* are both equal to zero and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpreted as either a segment counter (segment flow control) or a logical message counter (message flow control). In either case it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local ECL (see Chapter 4). An error is returned if the *drcnt* given increases the counter to more than 127.

ircnt

The interrupt request count byte (*ircnt*) is meaningful only when the low-order two bits of *lsflgs* = 01 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Thus, *ircnt* must always be 1 when the low-order two bits of *lsflgs* = 01 and has the effect of reenabling Interrupt Messages on the logical link. Note that Interrupt Messages can be reenabled only when the

previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call fails with an error (INTLCK) in R0.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The Interrupt/Link Service transmit queue for this logical link is full. Session Control is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and does not cause a Link Service Message to be queued when the condition clears. The program should retry the send after a 1 to 5 second delay. See Chapter 4 for a full discussion of flow control.
5	NOSUCH	The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed causes this error.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
18	BADFUO	One of several possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The low-order two bits of the *lsflgs* byte are both set to 1. o The low-order two bits of the *lsflgs* byte are 00 but the calling program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message. o The data request count byte (*drcnt*) increased the count of outstanding requests to more than 127 (decimal). o The low-order two bits of the *lsflgs* byte = 10, indicating a request for an Interrupt Message, but the *ircnt* argument is not equal to 1. o The low-order two bits of the *lsflgs* byte = 10, indicating a request for an Interrupt Message, but the outstanding requests counter for interrupts is already equal to one. o The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
19	INTLCK	The low-order two bits of *lsflgs* = 10, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example of NTLS requests five segments from the remote program over logical link 23. Assume that the calling program requested segment flow control in its NTCI or NTCC call for this link. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0   10.          ;DEFINE ARGUMENT BLOCK
           .
           .
           .
         .NTLS    #ARGBLK,#23.,#0,#5,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTLS    23.,0,5,,5
           .
           .
           .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Send Disconnect Message: NTDI (DECnet)

The NTDI call terminates an established logical link. Messages in the pending message queue remains but no more messages can be sent over the link. The User Link Address (ULA) is freed and can be used for another logical link. If the last Network Data Message sent did not have the end of message flag (bit 1 of the *dmflgs* argument) set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call fails with an error.

Successful completion of a NTDI call assures the calling program that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not, however, guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see the section "Send/Receive Disconnect Message" in Chapter 3). Otherwise, the NTDI call provides no particular advantage over a NTLA call (see the section "Send Link Abort Message") in terminating a logical link. You can send up to 16. bytes of user data with the Disconnect Message.

MACRO Format

```
.NTDI area,ula,bufptr,buflen,bufoff,rib
```

or

```
label:  $NTDI ula,bufptr,buflen,bufoff,rib
```

Argument Descriptions

label

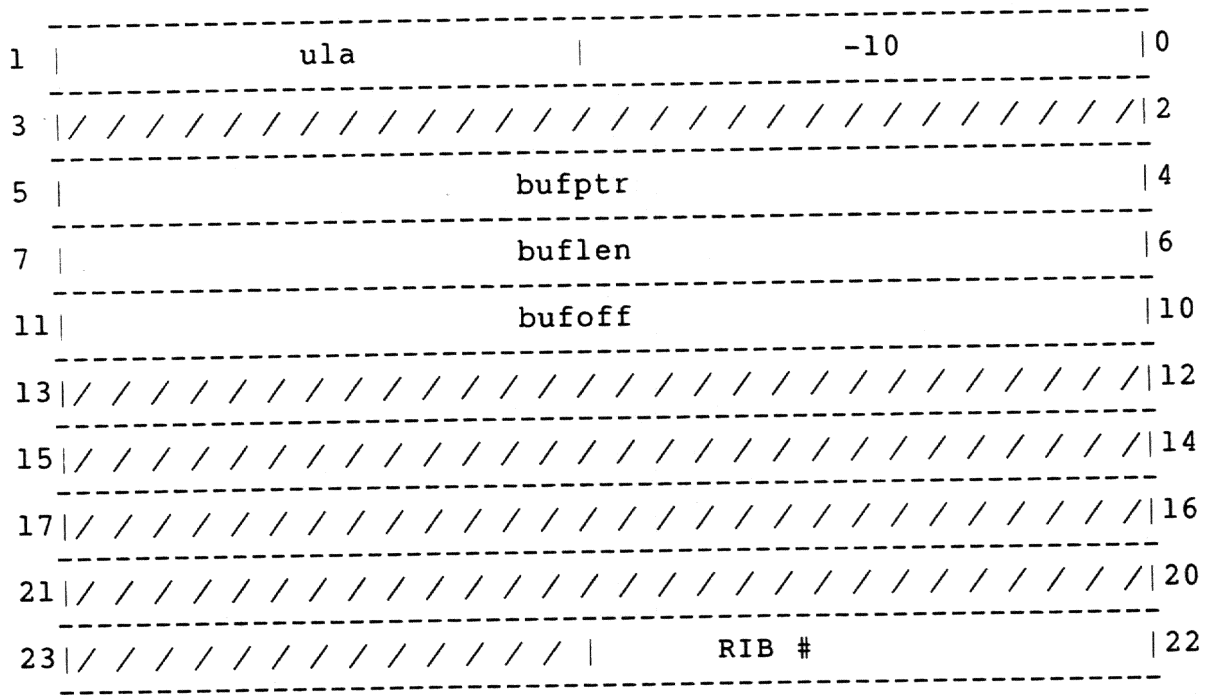
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

> The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTDI call, the argument block contains:



ula

The *ula* is a byte value specifying the User Link Address (1 to 255 decimal) chosen by this program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0. to 16.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16. and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The User Link Address (*ula*) does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed causes this error.
18	BADFUO	One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. Valid values range from 1 to 255 (1 to 377 octal).
19	INTLCK	The last segment sent over this logical link did not have the *eom* flag set. ECL does not permit the link to be disconnected unless all messages have been completely transmitted. Note that this test for the end of message flag is done, regardless of the flow control option chosen by the remote program.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet calls cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example disconnects logical link 255 and includes user data explaining the reason for the disconnect. Note that the remote program would have to be coded to recognize this message and take appropriate action. RIB #5 is used. For example:

```
ARGBLK:  .BLKW0    10.              ;DEFINE ARGUMENT BLOCK
MSGBUF:  .ASCII    /ALLDONE/
           .
           .
           .
         .NTDI     #ARGBLK,#255.,#MSGBUF,#7,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTDI     255.,MSGBUF,7,,5
MSGBUF:  .ASCII    /ALLDONE/
           .
           .
           .
         MOV       #ARGBLK,R0
         JSR       PC,$$$MSR
```

Send Link Abort Message: NTLA (DECnet)

The NTLA call terminates a logical link immediately. If there are still messages in the transmit queues waiting for acknowledgment from the remote ECL, the system discards them. These messages may or may not have been received by the remote system (see the section "Send/Receive Link Abort Message" in Chapter 3).

You can send up to 16. bytes of user data with the message. If you issue the NTLA call for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If you issue the NTLA call for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program.

MACRO Format

.NTLA *area,ula,bufptr,buflen,bufoff,rib*

or

label: $NTLA *ula,bufptr,buflen,bufoff,rib*

Argument Descriptions

label

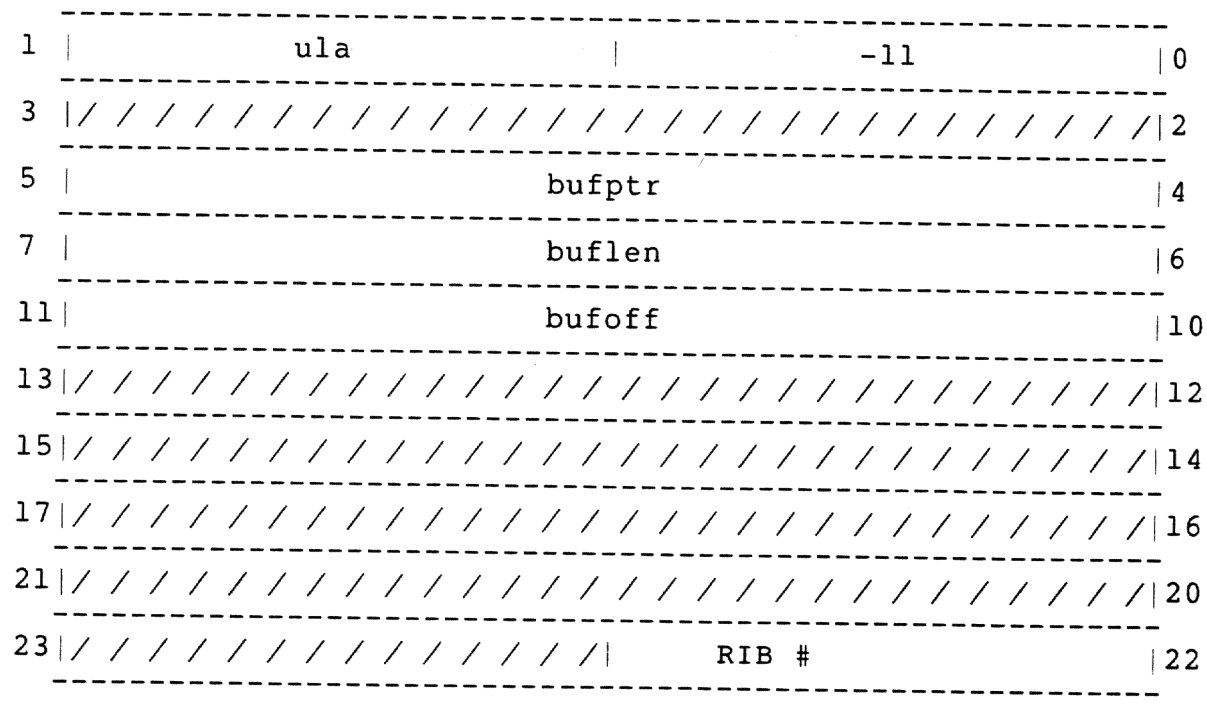
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTLA call, the argument block contains:



ula

The *ula* is a byte value specifying the User Link Address (1. to 255.) chosen by this program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

bufptr

This word defines the starting address where the optional user message data begins. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent from the buffer. Possible values for *buflen* range from 0. to 16.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

Note

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16. and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100. would be sent.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The User Link Address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the last case, a Disconnect or Link Abort Message has already been queued for this link.
18	BADFUO	One of two possibilities: o The job is not a declared receiver on the specified RIB number. Before any Receive can succeed on that RIB number, the job must be entered in the receiver list. o The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.

Possible Errors (Returned in R0) (Cont.)

Decimal Value	ERR.STB Mnemonic	Meaning
32	NOBUFS	System buffers are not currently available to store this message. A retry may succeed.
62	NORTS	DECnet/E has not been enabled. Normal DECnet cannot be executed until the system manager enables DECnet/E.
66	ERRERR	DECnet/E was not installed. The network functions cannot be executed.

Example

The following example aborts logical link 17. RIB #5 is used. No user data is sent so the buffer arguments are set to zero. For example:

```
ARGBLK:  .BLKW0   10.              ;DEFINE ARGUMENT BLOCK
           .
           .
           .
         .NTLA    #ARGBLK,#17.,#0,#0,#0,#5
```

The following example shows the "$" form of the same call:

```
ARGBLK:  $NTLA    17.,,,,,5
           .
           .
           .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Receive: MRCV (Local and DECnet)

The MRCV call retrieves a message from the calling program's pending message queue. It returns control parameters to the FIRQB and XRB, and user data, if any, to a buffer defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer. The user data can be retrieved all at once or in portions, and portions can be discarded. The formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message" show what is returned to the FIRQB and XRB for the various types of messages.

The Receive call can be selected to get:

- The first message in the queue
- The first message in the queue from either a network or a local sender
- The first message in the queue for a particular logical link

You can set a "sleep" flag and associated timer to indicate that the calling program is to wait if there are no appropriate messages pending in the queue. Otherwise, if there are no appropriate messages pending, the MRCV call terminates with an error (NOSUCH).

MACRO Format

```
.MRCV area,rmod,sndr,qual,bufptr,buflen,bufoff,slptim,rib
```

or

```
label:  $MRCV rmod,sndr,qual,bufptr,buflen,bufoff,slptim,rib
```

Argument Descriptions

label

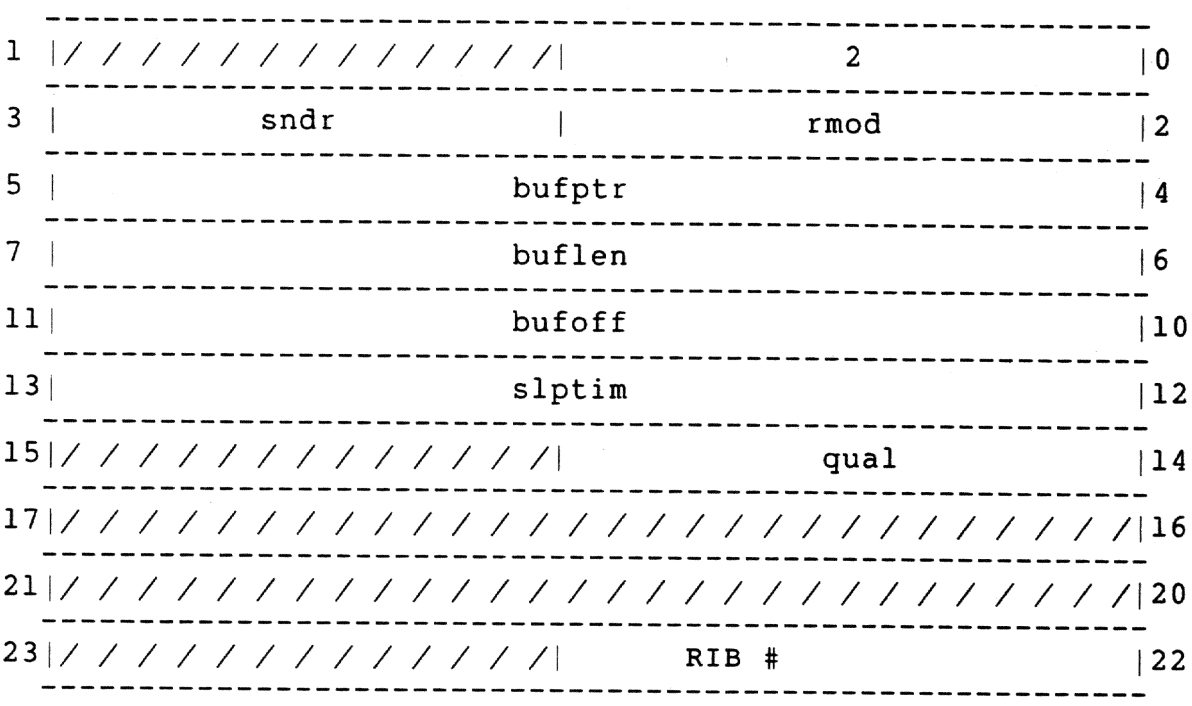
The *label* used for the "$" form of the call defines the starting address of the 10-word argument block created by the "$" form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the "$" form.

Note

The "." form fills in only the arguments specified in the call (see the section "Programming Background"). If you omit an argument, the corresponding field in the argument block is left alone. The "$" form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MRCV call, the argument block contains:



rmod

The low-order four bits of this byte define the way the Receive call is executed:

bit 0 = 1 The program is to be put into a sleep state if no appropriate message is pending in the queue. The *slptime* parameter regulates the time that the program sleeps. The conditions under which the program is awakened are discussed under *slptime*.

= 0 The program should not be put into a sleep state if there is no appropriate pending message in the queue. The call terminates immediately with a NOSUCH error to inform the program that no appropriate messages are pending.

bit 1 = 1 Any excess data that does not fit in the specified buffer is to be discarded. Data could be left over if the value of *buflen* is smaller than the amount of user data in the message.

= 0 Excess data that does not fit in the buffer specified in this call is to be kept for retrieval on later Receive calls.

bit 2 = 1 A message is to be selected from a local sender. The selection can be further qualified with the *sndr* and *qual* arguments.

= 0 Local selection is not requested. If both bit 2 and bit 3 are zero, the first message in the queue is returned.

bit 3 = 1 A message is to be selected from a network sender. The selection can be further qualified with the *sndr* and *qual* arguments.

= 0 Network selection is not requested. If both bit 3 and bit 2 are zero, the first message in the queue is returned. If both bit 3 and bit 2 are one, local selection prevails.

One way to document and set these flags is to assign mnemonics to the values that set the bits. For example:

```
SLP    = 1
TRUNC  = 2
LOCAL  = 4
NET    = 8
```

To request a network message and the sleep option, then, the *rmod* argument could be coded as #NET+SLP.

sndr

The sender selection byte (*sndr*) selects a particular local sender or a particular logical link for this Receive call.

If bit 2 of *rmod* = 1 (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If bit 2 of *rmod* = 0 and bit 3 = 1, a nonzero value for *sndr* is interpreted as a ULA. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either bit 2 or bit 3 of *rmod* = 1, the *qual* argument has special meaning. The *sndr* and *qual* arguments are ignored if both bit 2 and bit 3 of *rmod* are zero.

qual

The sender selection qualifier (*qual*) byte is normally zero for user applications.

If bit 2 of *rmod* = 1 (local selection) and *sndr* = 0, then any nonzero value for *qual* is a special Receive call requesting a message from the system (job number 0). This special case is used by the RSTS/E utility ERRCPY, which processes messages from the monitor error logging routines.

The *qual* argument is ignored if both bit 2 and bit 3 of *rmod* = 0 or if *sndr* = 0.

Table 11-4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 11-4: Sender Selection Summary

rmod				
bit 3	*bit 2*	*sndr*	*qual*	Result
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	1	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times 2. Only messages from that job are delivered on this MRCV call.

Table 11-4: Sender Selection Summary (Cont.)

rmod bit 3	*rmod* bit 2	*sndr*	*qual*	Result
1	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a user link address. Only network messages from the designated logical link are delivered on this MRCV call.

bufptr

This word defines the starting address where the user message data is to be delivered. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the maximum number of bytes to be delivered to the buffer on this Receive call.

If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (bit 1 of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, then only the amount indicated by *buflen* is returned. The rest is saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining the word at FIRQB+12 in the data returned by the Receive call. The number of bytes actually returned to the buffer is returned in the word at XRB+2. See the formats in the sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

Note

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100. and a *bufoff* of 20. Up to 100. bytes beginning at *bufptr*+20. could be delivered on the Receive call.

slptim

The *slptime* argument is significant only when bit 0 of *rmod* = 1. In that case, *slptime* defines the length of time, in seconds, that a program is to sleep when no appropriate message is in the queue. The program sleeps until:

- The sleep timer (*slptime*) expires.
- Any new message is placed in the queue (not just an appropriate one).
- A delimiter is typed on a terminal opened by the job or assigned to the job.
- Log-ins are disabled. (This could occur if the system is being shut down).
- A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read, or has entered an input wait state.)
- The job itself has opened the XM: (DMC11 driver) and a message is received for the job.
- The system time is changed.

The program is awakened in all cases with an error (NOSUCH) but is not passed as a message. To obtain a pending message the program must execute another Receive call. Since the program can be awakened by any of the conditions listed, a check for pending messages can be made by executing a MRCV without a sleep.

rib

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs) numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call.

Possible Errors (Returned in R0)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	For a Receive call without sleep (bit 0 in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (bit 0 in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV call to retrieve any pending messages.
18	BADFUO	No RIB exists for this program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example

In the following example a Receive call is issued for network messages from logical link 25. The sleep bit is set and the timer is set to five seconds. The truncate bit is 0 so all the information that is available to be returned to the buffer is saved for another Receive call. RIB #5 is used. Note that the program examines the byte at FIRQB+4 to see what type of message is available and jumps to an appropriate section of code depending upon its value. For example:

```
ARGBLK:  .BLKW0   10.          ;DEFINE ARGUMENT BLOCK
           .
           .
           .
         .MRCV    #ARGBLK,#9.,#25.,#0,,#0,#0,#5,#5
```

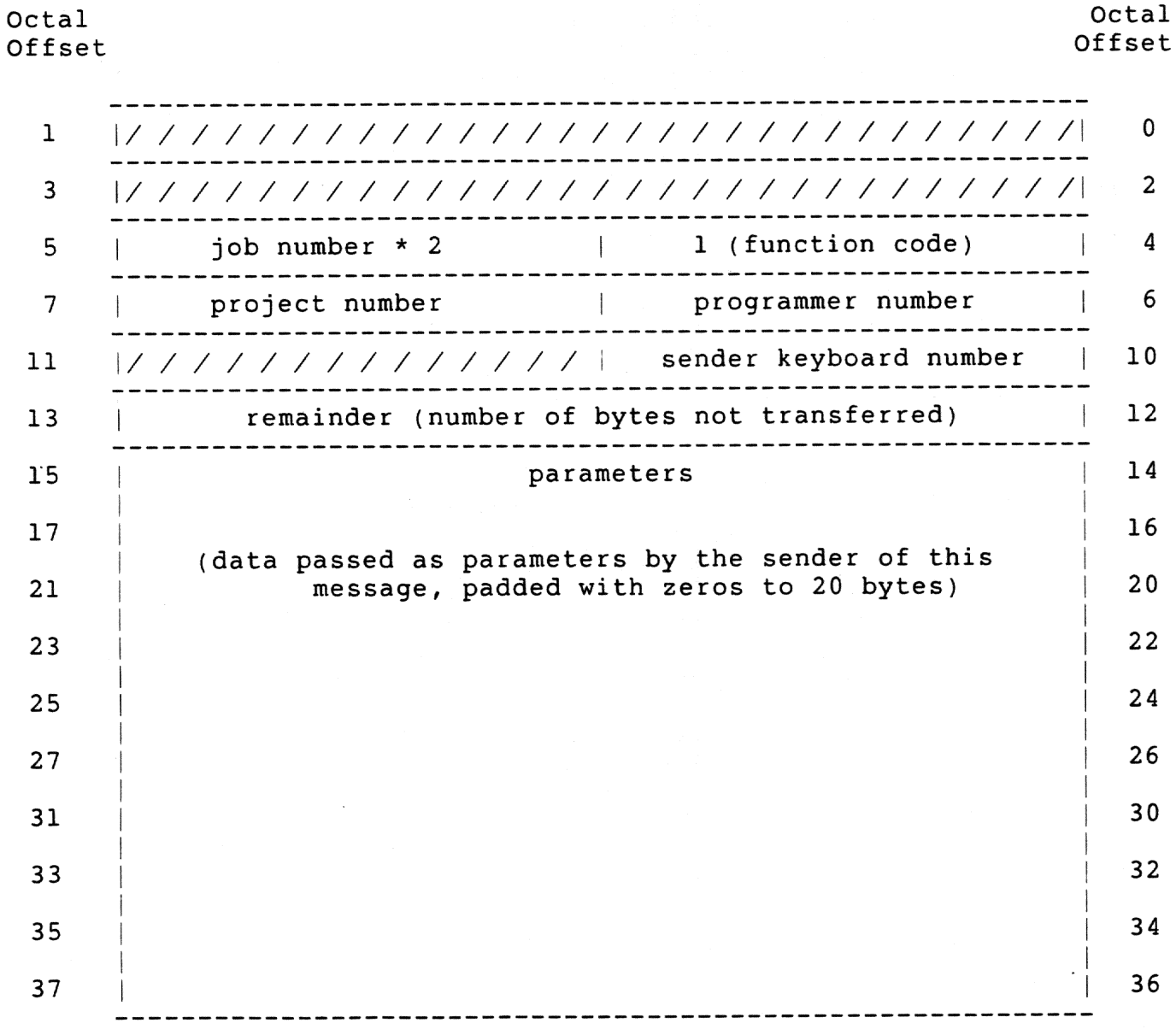
The following example shows the "$" form of the same call:

```
ARGBLK:  $MRCV    9.,25.,0,0,0,5,5
           .
           .
           .
         MOV      #ARGBLK,R0
         JSR      PC,$$$MSR
```

Format of Received Local Data Message

The following format describes the information returned to the FIRQB and XRB for a Local Data Message. Up to 512 (decimal) bytes of user data can be delivered to the buffer specified in the Receive (MRCV) call.

FIRQB



FIRQB+4 The function code for a Local Data Message.

FIRQB+5 Two times the job number of the local sender.

FIRQB+6 The PPN of the local sender.

FIRQB+10 The keyboard number of the local sender or 377 if the sender is detached.

FIRQB+12 The number of bytes of user data not delivered to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument in the MRCV call.

FIRQB+14 The data passed as parameters by the sender of this message is returned starting at this address. The system pads any unused bytes with zeros to a length of 20 bytes.

XRB

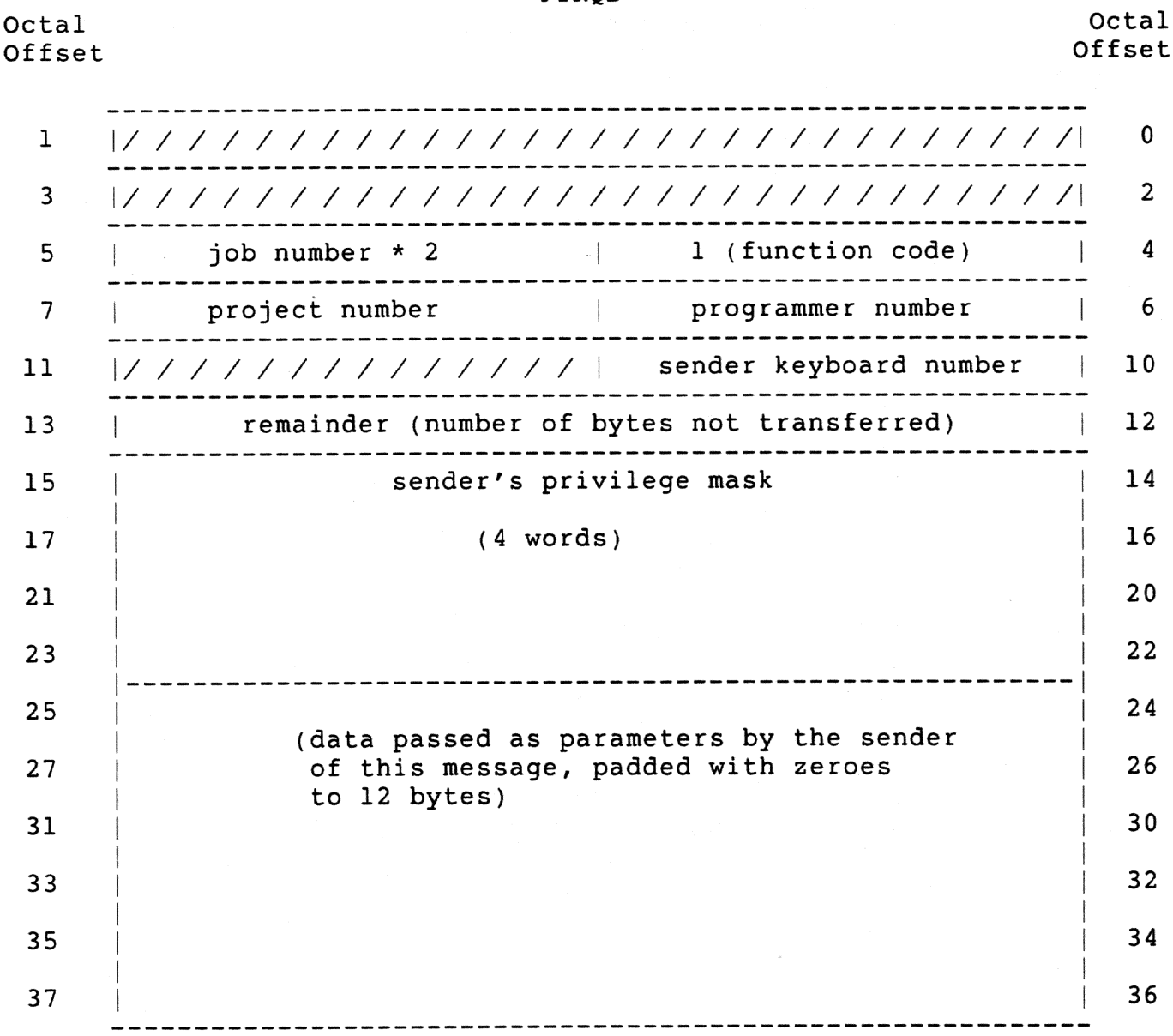
Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Local Data Message with Privilege Mask

The following format describes the information returned to the FIRQB and XRB for a Local Data Message with Privilege Mask. Up to 512 (decimal) bytes of user data can be delivered to the buffer specified in the Receive (MRPR) call.

FIRQB



FIRQB+4 The function code for a Local Data Message.

FIRQB+5 Two times the job number of the local sender.

FIRQB+6 The PPN of the local sender.

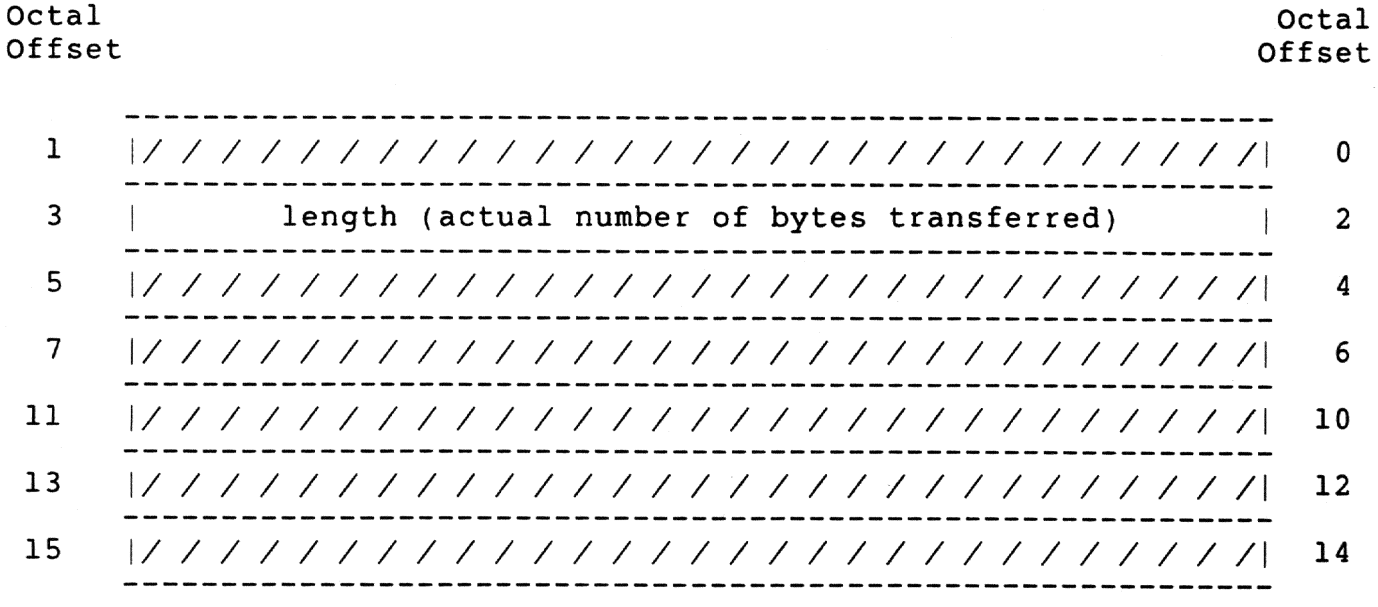
FIRQB+10 The keyboard number of the local sender or 377 if the sender is detached.

FIRQB+12 The number of bytes of user data not delivered to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument in the MRPR call.

FIRQB+14 The sender's privilege mask.

FIRQB+20 The data passed as parameters by the sender of this message is returned starting at this address. The system pads any unused bytes with zeros to a length of 12 bytes.

XRB



XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Connect Initiate Message

The following format describes the information returned to the FIRQB and XRB for a received Connect Initiate Message. A 120. byte Connect Data Block and up to 16. bytes of user data are also returned to the buffer specified in the Receive call. Figure 11-2 shows the format of a received Connect Data Block; Table 11-5 describes it in detail.

FIRQB

Octal Offset		Octal Offset
1	/ /	0
3	/ /	2
5	/ / / / / / / / / / / / / / / / | 2 (function code)	4
7	11a (local link address)	6
11	/ /	10
13	remainder (number of bytes not transferred)	12
15	/ /	14
17	/ / / / / / / / / / / / / / / / | rlmod: 0=no, 1=sg, 2=ms	16
21	rmax (maximum bytes received in 1 network data message)	20
23	tmax (maximum bytes sent in 1 network data message)	22
25	/ /	24
27	/ /	26
31	/ /	30
33	/ /	32
35	/ /	34
37	/ /	36

FIRQB+4 The function code for a Connect Initiate Message.

FIRQB+6 Local Link Address. Identifies the link for the receiving program's subsequent Connect Confirm or Connect Reject Message.

FIRQB+10 — The number of bytes of data not returned to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument in the Receive call.

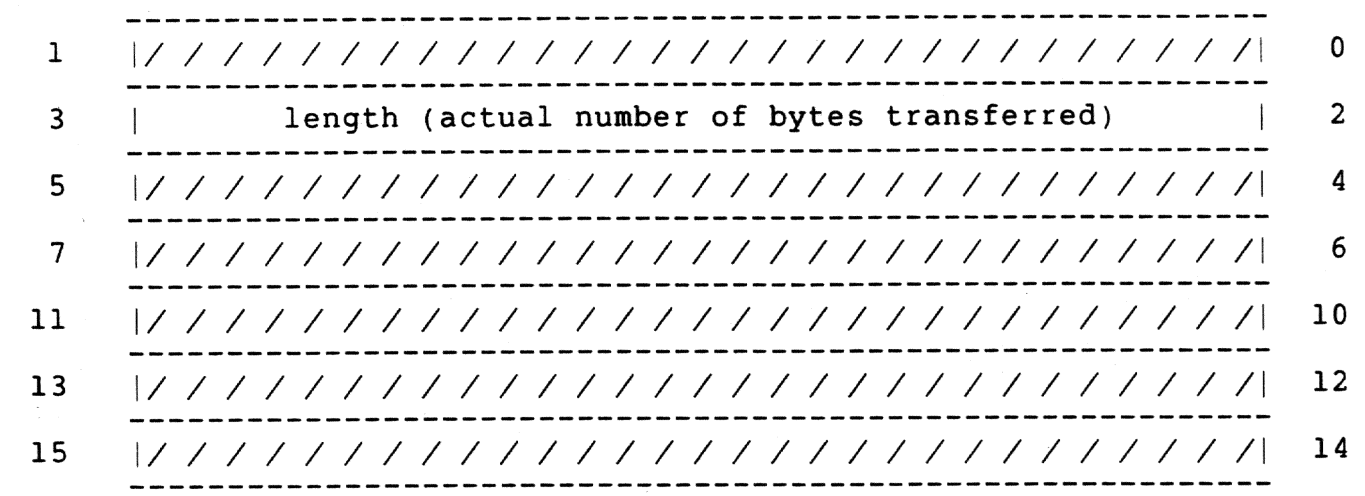
FIRQB+16 — Remote link modifier. Indicates the flow control in effect for the remote program:

0 = No flow control
1 = Segment flow control
2 = Message flow control

FIRQB+20 — Receive maximum, in bytes. The maximum amount of user data that is transmitted to the receiving program in one Network Data Message over this link. This value is supplied by the local ECL and is determined by the size of the receive buffers that it allocates. The maximum size of the receive buffers is set for ECL by Network Management. The receiving program can further limit the amount of user data to be received by specifying a smaller rmax argument in the Connect Confirm Message that accepts this request for a connection (see the section "Send Connect Confirm Message").

FIRQB+22 — Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the receiving program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.

XRB



XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

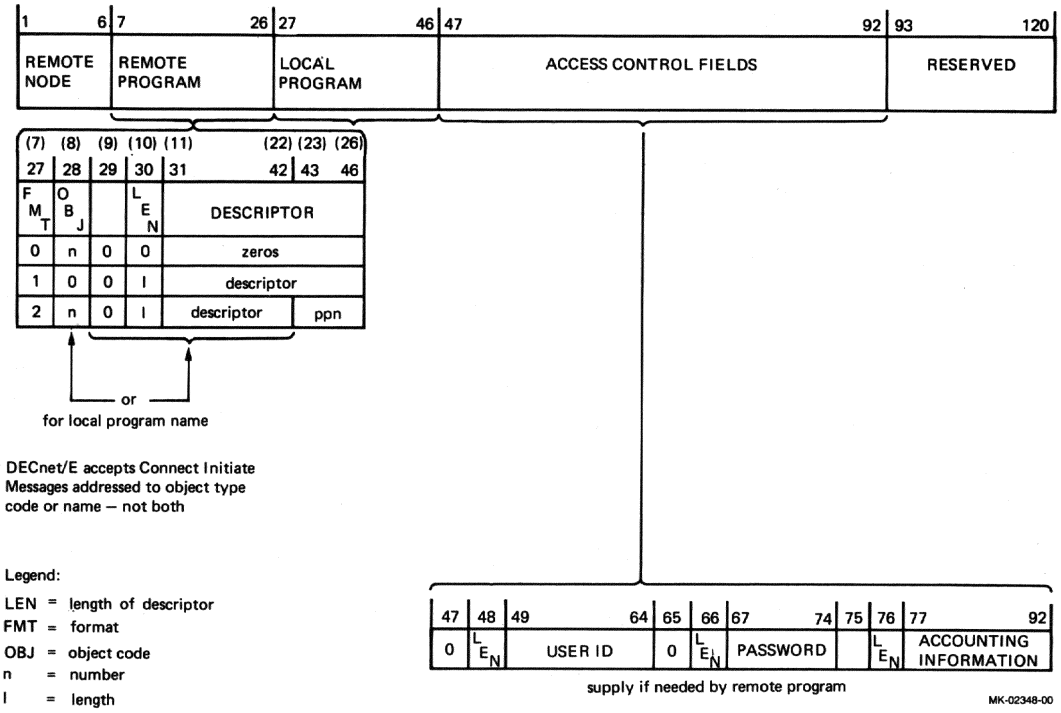


Figure 11-2: Format of Received Connect Data Block

Table 11-5: Format of Received Connect Data Block

Bytes	Content
1-6	**Remote Node.** Contains 1 to 6 characters naming the remote node from which the Connect Initiate Message was received. Names shorter than 6 characters are left-justified with space fill to 6 characters.
7-26	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type (general function) or a specific entity (program, task, or process). This identification is given in one of the following three formats.

Table 11-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 0 The remote program is identified by object type code alone. The fields are: **Bytes** — **Content** 7 — Zero, to indicate a format 0 name. 8 — Object type code of the remote program. 9-26 — Zero (not used).
	Format 1 The remote program is identified by a descriptor. The subfields are: **Bytes** — **Content** 7 — One, to indicate a format 1. 8 — Normally zero. This byte is usually zero unless the target DECnet system identifies the remote program with both name and object type code on outgoing connect requests. 9 — Reserved. 10 — Length (in bytes) of the descriptor, 0 to 16. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-26). 11-26 — Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations.

Table 11-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
	Format 2 This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. The group and user codes refer to what is called the project-programmer number (PPN) on some DIGITAL systems, including RSTS/E. Other DIGITAL systems may refer to these codes as the user identification code, or UIC. DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Bytes	Content
7	Two, to indicate a format 2.
8	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
9	Reserved.
10	Length (in bytes) of descriptor, 0 to 12. Gives the number of characters to be interpreted as the program descriptor in the following field (bytes 11-22).
11-22	Remote program descriptor, if this is used instead of object type code. The descriptor is left-justified ASCII characters.
23-24	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, byte 23 is the project number in a RSTS/E PPN.
25-26	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, byte 25 is the programmer number in a RSTS/E PPN.

Table 11-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
27-46	**Local Program Name.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are:

Bytes	Content
27	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address the local program.
28	Object type code, if used by the remote program to address the local program (formats 0 or 2).
29	Always zero.
30	Length of name, in bytes. Equal to 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
31-36	Local program name, if used by the remote program to address the local program (formats 1 or 2).
37-42	Always zero.
43-44	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.
45-46	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by ECL to identify the local program but passed on here.

Table 11-5: Format of Received Connect Data Block (Cont.)

Bytes	Content
47-92	**Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The local program could use this information to log into the RSTS/E system, grant classes of privileges, or whatever. The most precise checking done in a normal login on DIGITAL systems involves user identification, password, and account number, so three fields are defined here for those items. However, DECnet does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system passes this information on to the receiving program. Note Normally, DECnet/E handles the access control fields passed in a Connect Initiate Message and creates a logged in job. However, to be compatible with DECnet/E V2.0 and earlier, NCP may define an object as a "Verification Program" that creates a logged out job and passes the access control data.

Bytes	Content
47	Reserved.
48	Length of user identification information specified in the next field, 0 to 16 bytes.
49-64	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the PPN at login.
65	Reserved.
66	Length of password information specified in the next field, 0 to 8 bytes.
67-74	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the PPN used at login.

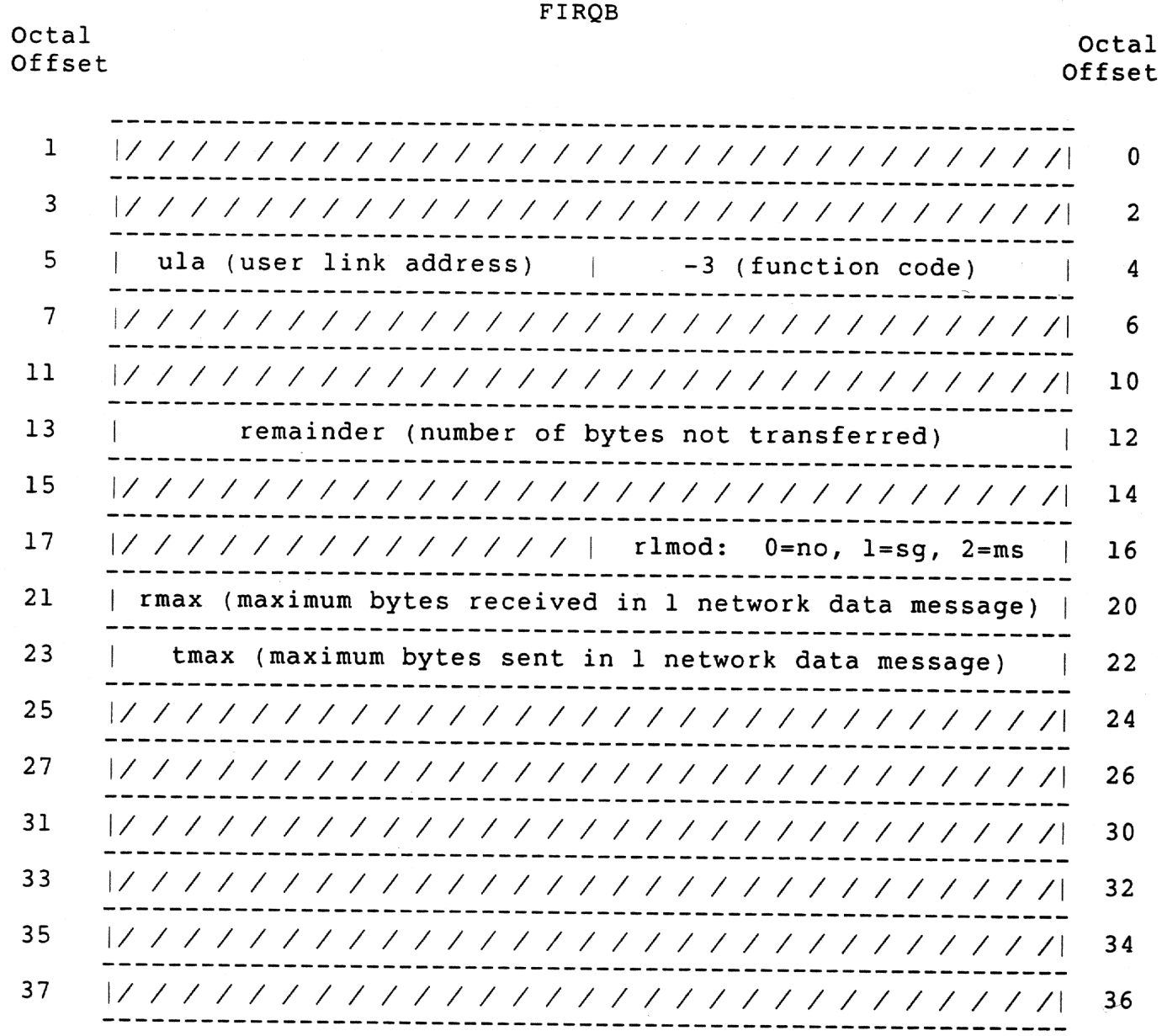
Table 11-5: Format of Received Connect Data Block (Cont.)

Bytes	Content	
	75	Reserved.
	76	Length of the accounting information specified in the following field, 0 to 16 bytes.
	77-92	Account number or system password. If the remote system is a DECnet/E system, the system password is passed here. If it is not a DECnet/E system, see the remote operating system's documentation for more information.
93-120	Reserved.	

Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the local program. The link is identified by the User Link Address specified by the local program in the Connect Initiate Message. The following format describes the information returned to the FIRQB and XRB. Up to 16 bytes of user data can also be returned to the buffer specified in the Receive call.

FIRQB



FIRQB+4 The function code for a Connect Confirm Message.

FIRQB+5 — User Link Address, previously defined in the receiving program's Connect Initiate Message.

FIRQB+12 — Number of bytes of data not returned to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument for this Receive call.

FIRQB+16 — Remote link modifier. Indicates the flow control in effect for the remote program:

0 = No flow control
1 = Segment flow control
2 = Message flow control

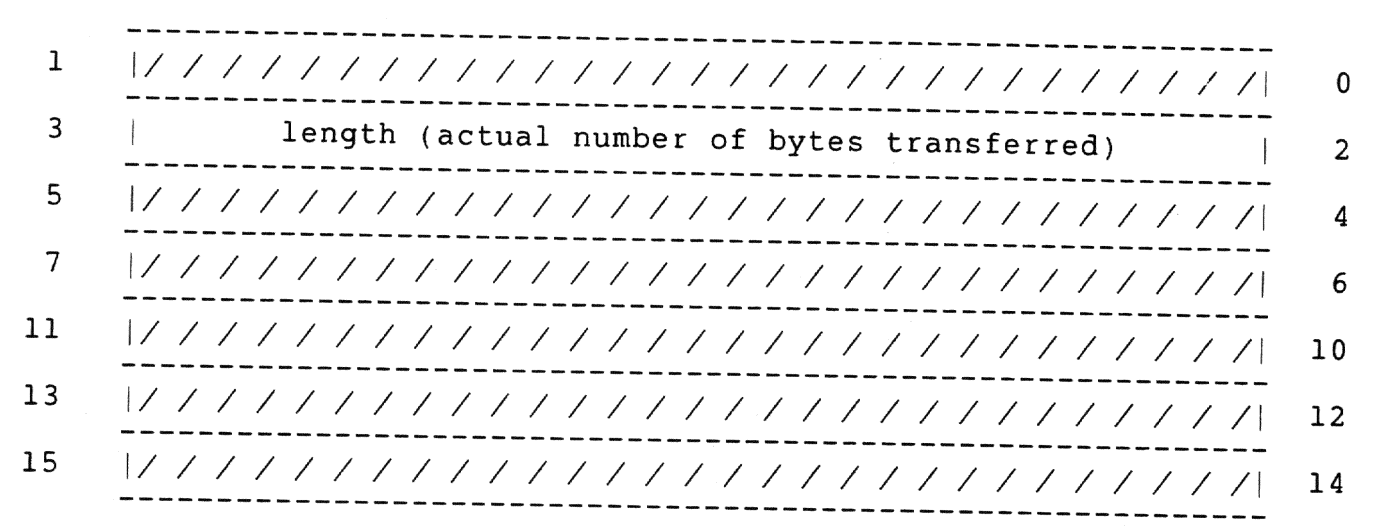
FIRQB+20 — Receive maximum, in bytes. The maximum amount of user data that is transmitted to the local program in one Network Data Message over this link. This is:

- Equal to the *rmax* value specified by the local program in the Connect Initiate Message for this link
- A lesser value supplied by the local ECL, determined from the maximum size of the receive buffers it allocates.

FIRQB+22 — Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote ECL. The local ECL enforces this limit for all Network Data Messages sent by the local program on this link.

Format of Received Connect Confirm Message MACRO

XRB

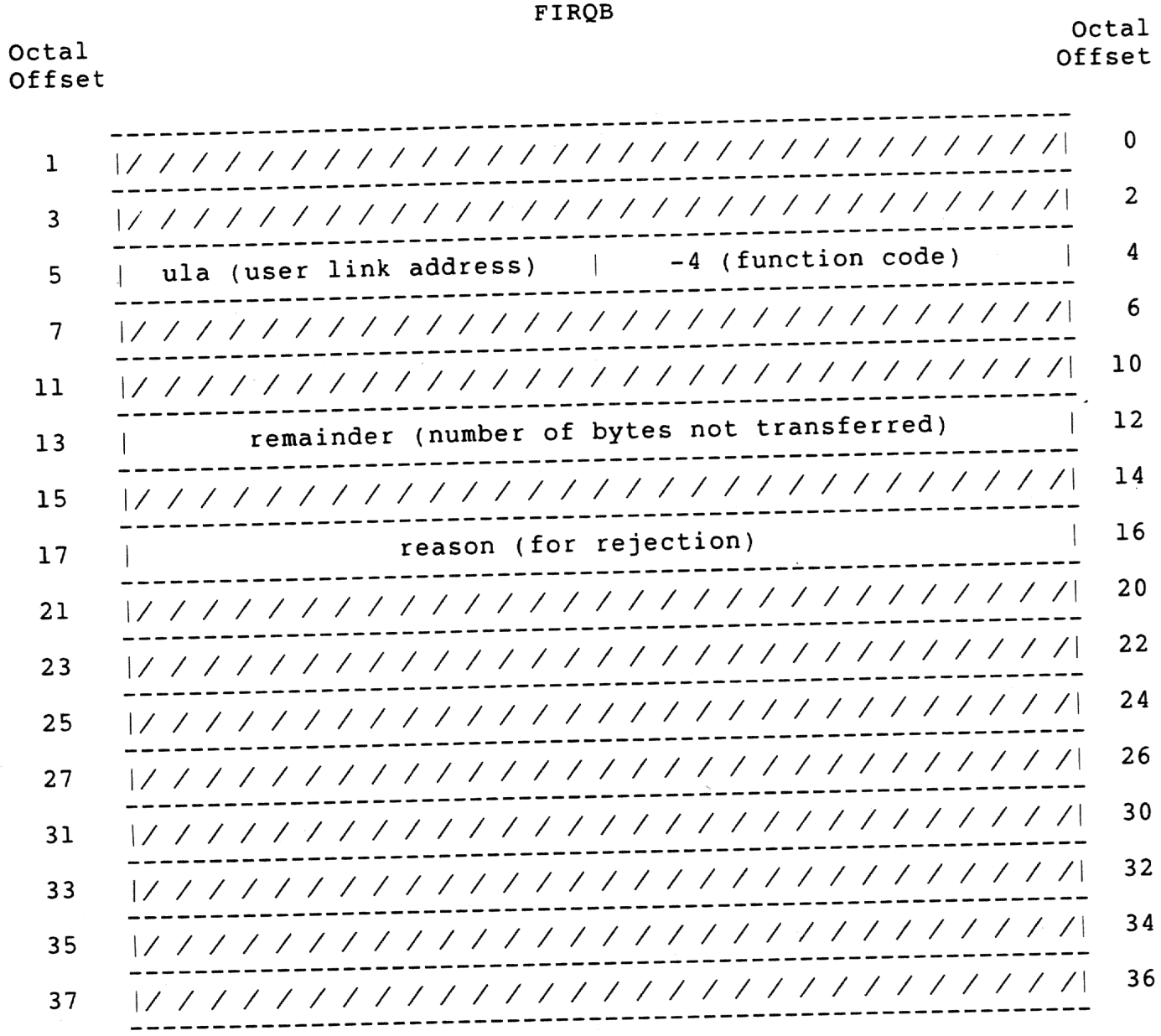


XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the receiving program's request for a logical link. It can be from the remote program or from the local or remote ECL. The link is identified by the User Link Address (ULA) previously established in the receiving program's Connect Initiate Message. The following format describes the information returned to the FIRQB and XRB. Rejections from ECL contain a reason code explaining the reason for the rejection and no user data. Rejections from the remote program can contain up to 16. bytes of user data delivered to the buffer defined in the Receive call.

FIRQB



Format of Received Connect Reject Message
MACRO

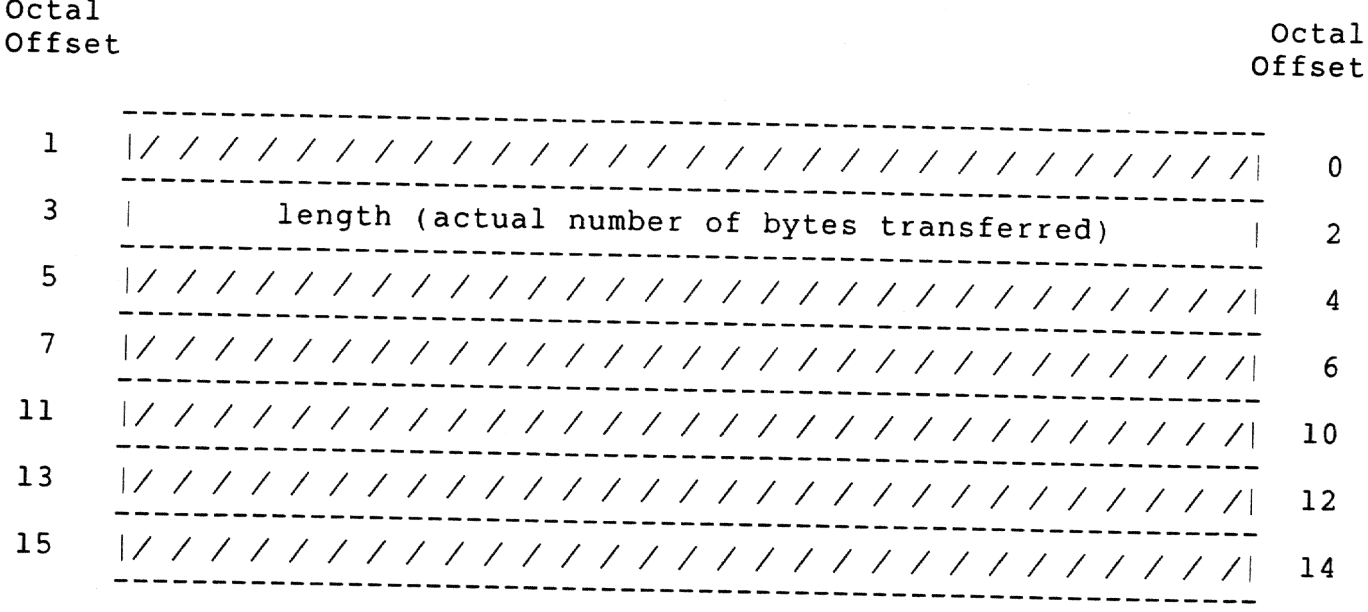
FIRQB+4 — Function code for Connect Reject Message.

FIRQB+5 — User Link Address, previously defined in the receiving program's Connect Initiate Message.

FIRQB+12 — Number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on the Receive call.

FIRQB+16 — Identifies the reason for the rejection (see Appendix B).

XRB

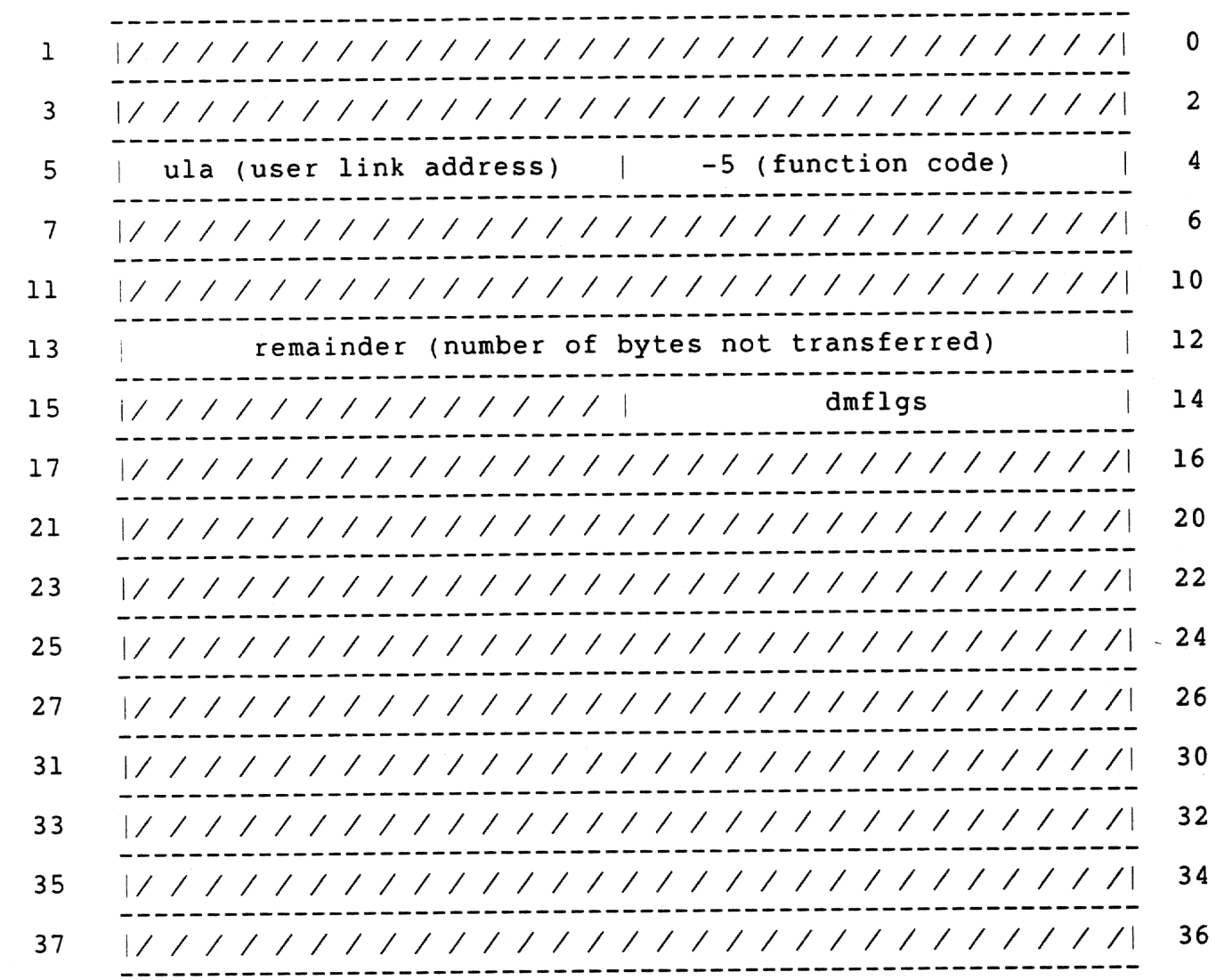


XRB+2 — The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Network Data Message

A received Network Data Message contains user data from a remote program. The following format describes the information returned to the FIRQB and XRB. The user data accompanying the Network Data Message is delivered to the buffer specified in the Receive call.

FIRQB



FIRQB+4 Function code of Network Data Message.

FIRQB+5 User Link Address, previously defined in this program's Connect Initiate or Connect Confirm Message.

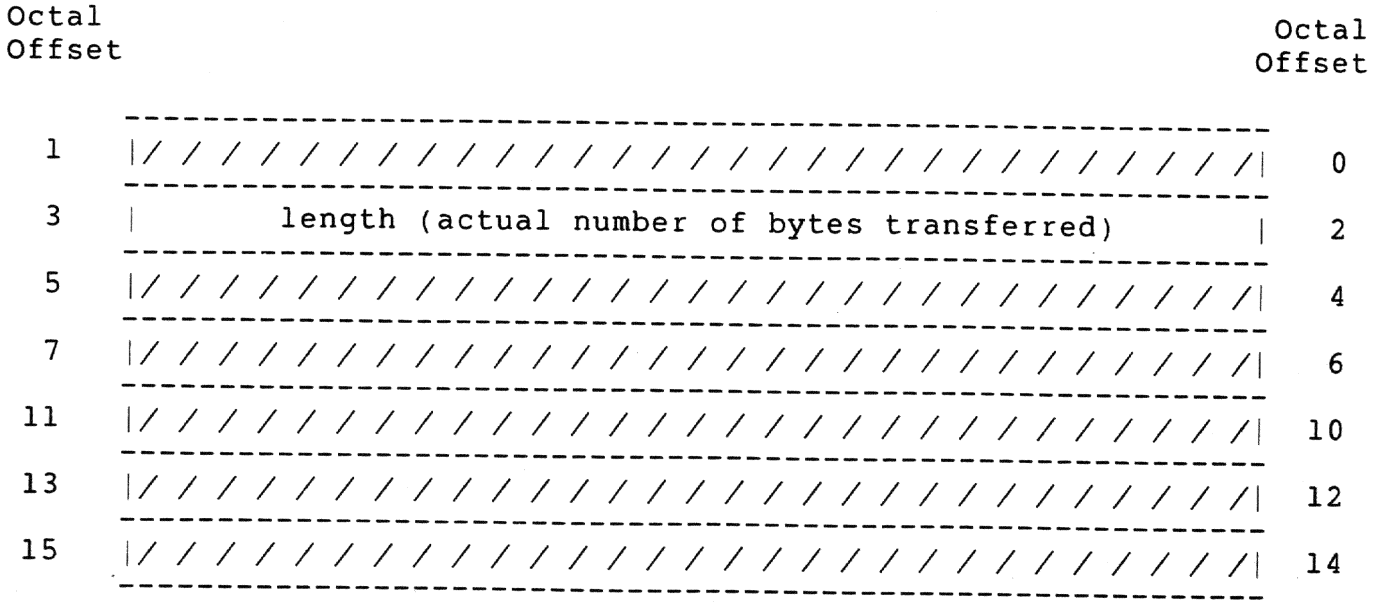
FIRQB+12 Number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on the Receive call.

FIRQB+14 Data message flags. The low-order two bits of this byte indicate whether this is the beginning, middle, end, or sole segment of a logical message.

Bit 1	Bit 0	Meaning
0	0	Middle segment of a message
0	1	First segment of a message
1	0	Last segment of a message
1	1	Sole segment of a message

If this program requested message flow control when the logical link was established, DECnet/E uses bit 1 to determine the end of a logical message. It does not use bit 0 but passes it on to the receiving program.

XRB

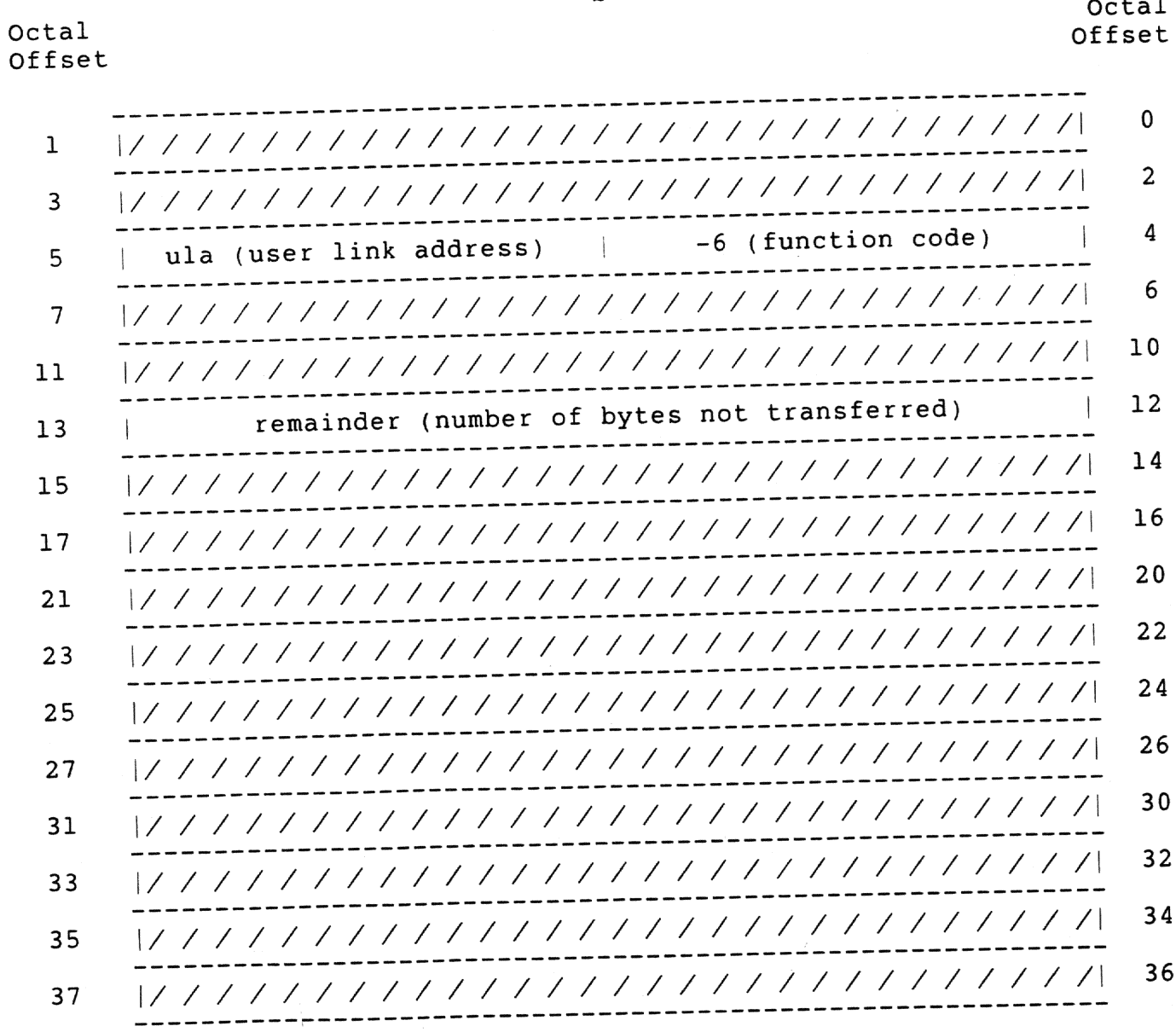


XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Interrupt Message

The local ECL places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The following format describes the data returned to the FIRQB and XRB. Up to 16. bytes of user data can be delivered to the buffer specified in the Receive call.

FIRQB

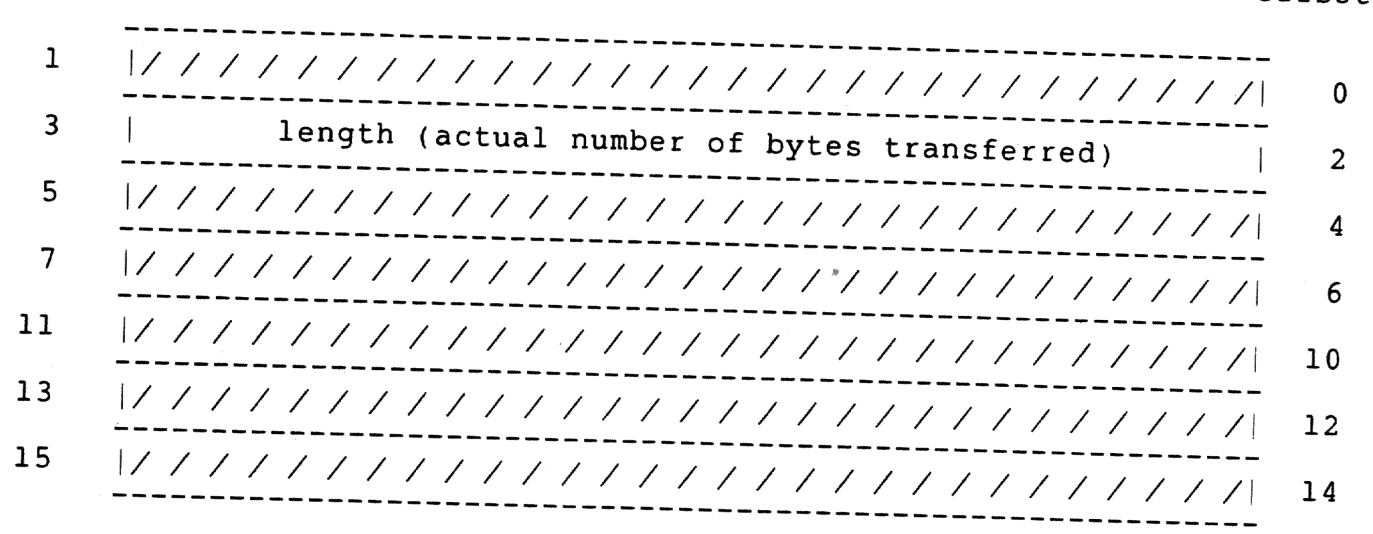


FIRQB+4 Function code for an Interrupt Message.

FIRQB+5 User Link Address, previously defined by this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 The number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on this Receive call.

XRB



XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call when the pending message queue has emptied after a condition has cleared that previously inhibited this program from sending (see the section "Flow Control" in Chapter 4). Status information (in the same format as a Link Service Message) can also be returned to the FIRQB and XRB on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The following format describes the information returned. No user data is delivered to the buffer with a Link Service Message.

FIRQB

Octal Offset		Octal Offset
1	/ /	0
3	/ /	2
5	ula (user link address) | -7 (function code)	4
7	/ /	6
11	/ /	10
13	(always = 0)	12
15	rstat (remote status) | lstat (local status)	14
17	lir (local interrupt cnt) | ldr (local data count)	16
21	rir (remote interrupt cnt) | rdr (remote data count)	20
23	dtc (data xmit que cnt) | dtm (data xmit que max)	22
25	ltc (lnk srvc xmit que cnt) | ltm (lnk srvc xmit que max)	24
27	mcnt (message count) | mmax (message max)	26
31	/ / / / / / / / / / / / / / / | mula (messages for link)	30
33	/ /	32
35	/ /	34
37	/ /	36

Format of Received Link Service Message
MACRO

FIRQB+4 — Function code for Link Service Message.

FIRQB+5 — User Link Address, previously defined by this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 — Always zero for Link Service Messages. (This word is the remainder field in other returned messages.)

FIRQB+14 — Local status flags. The two low-order bits give the status of outgoing messages:

Bit 0 = 1 — Outgoing Network Data Messages are inhibited. A Link Service Message is delivered when the condition clears.

= 0 — No restrictions on sending Network Data Messages.

Bit 1 = 1 — Outgoing Interrupt and Link Service Messages are inhibited. A Link Service Message is delivered when the condition clears.

The two high-order bits give the status of incoming messages:

Bit 7	Bit 6	Meaning
0	0	Incoming Data Message flow is on.
0	1	(Decimal value = 64) Incoming Data Message flow is on but the local ECL turns this link off if another Data Message is received before this program's pending message queue is emptied.
1	0	(Decimal value = 128) Incoming Data Message flow has been turned off by the local ECL. This program's pending message queue must be emptied before the link is turned on.
1	1	(Decimal value = 192) Incoming Data Message flow has been turned off by the local ECL but flow is scheduled to turn on as soon as system buffers become available.

FIRQB+15 Remote status flag.

Bit 7 = 1 (Decimal value = 128) The link has been turned off by the remote system due to a backpressure condition. No Network Data Messages can be sent until the remote system turns the link back on.

FIRQB+16 Local data request count: The count of segments or logical messages currently requested by this program, reflecting the actual number of segments or logical messages requested but not yet received.

FIRQB+17 Local interrupt request count: The count of Interrupt Messages currently requested by this program, reflecting the actual count of interrupts requested but not yet received. This count never exceeds one.

FIRQB+20 Remote data request count: The count of segments or logical messages currently requested by the remote program, reflecting the actual count of segments or logical messages requested but not yet sent.

FIRQB+21 Remote interrupt request count: The count of Interrupt Messages currently requested by the remote end of the logical link, reflecting the actual count of interrupts requested but not yet sent.

FIRQB+22 Data transmit queue maximum: The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.

FIRQB+23 Data transmit queue count: The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote ECL.

FIRQB+24 Interrupt/Link Service transmit queue maximum: The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when DECnet/E is enabled. It applies individually to all active logical links.

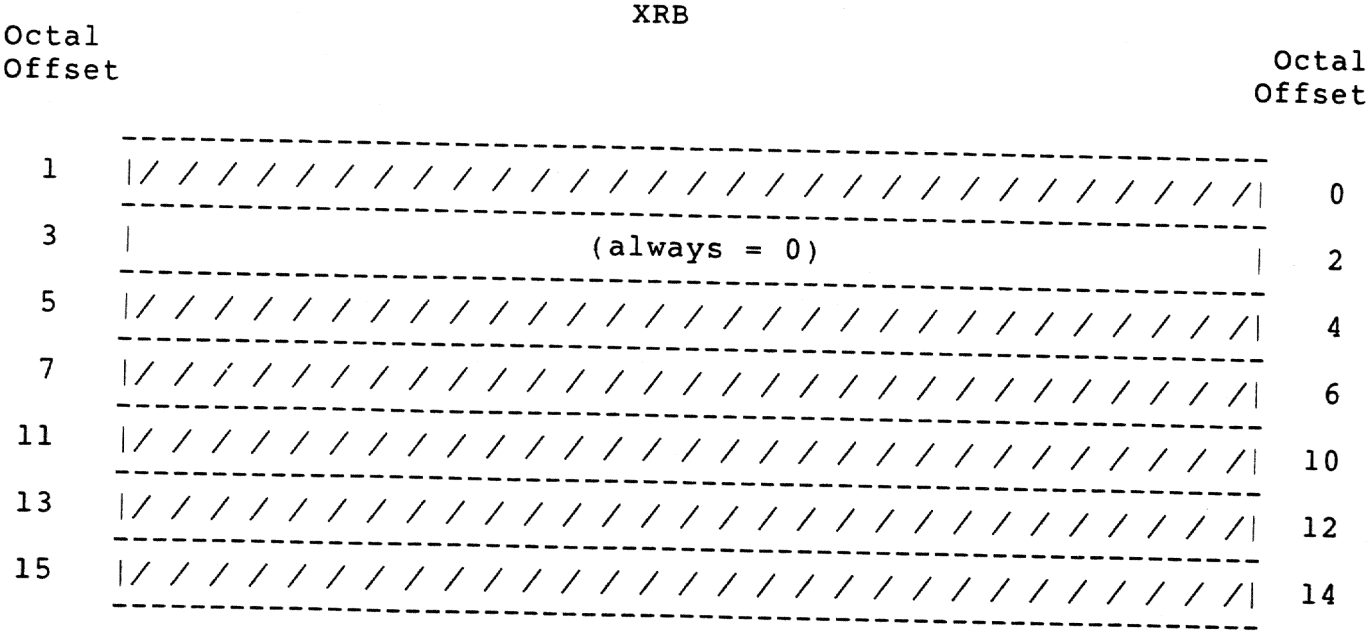
FIRQB+25 — Interrupt/Link Service transmit queue count: The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote ECL.

FIRQB+26 — Message maximum: The maximum number of incoming messages that are queued for this program. This maximum is set by this program in the Declare Receiver call.

FIRQB+27 — Message count: The total number of messages currently in the pending message queue for this program.

FIRQB+30 — Message count for this logical link: The number of messages currently in the pending message queue for this logical link, as identified by the user link address (FIRQB+5).

XRB

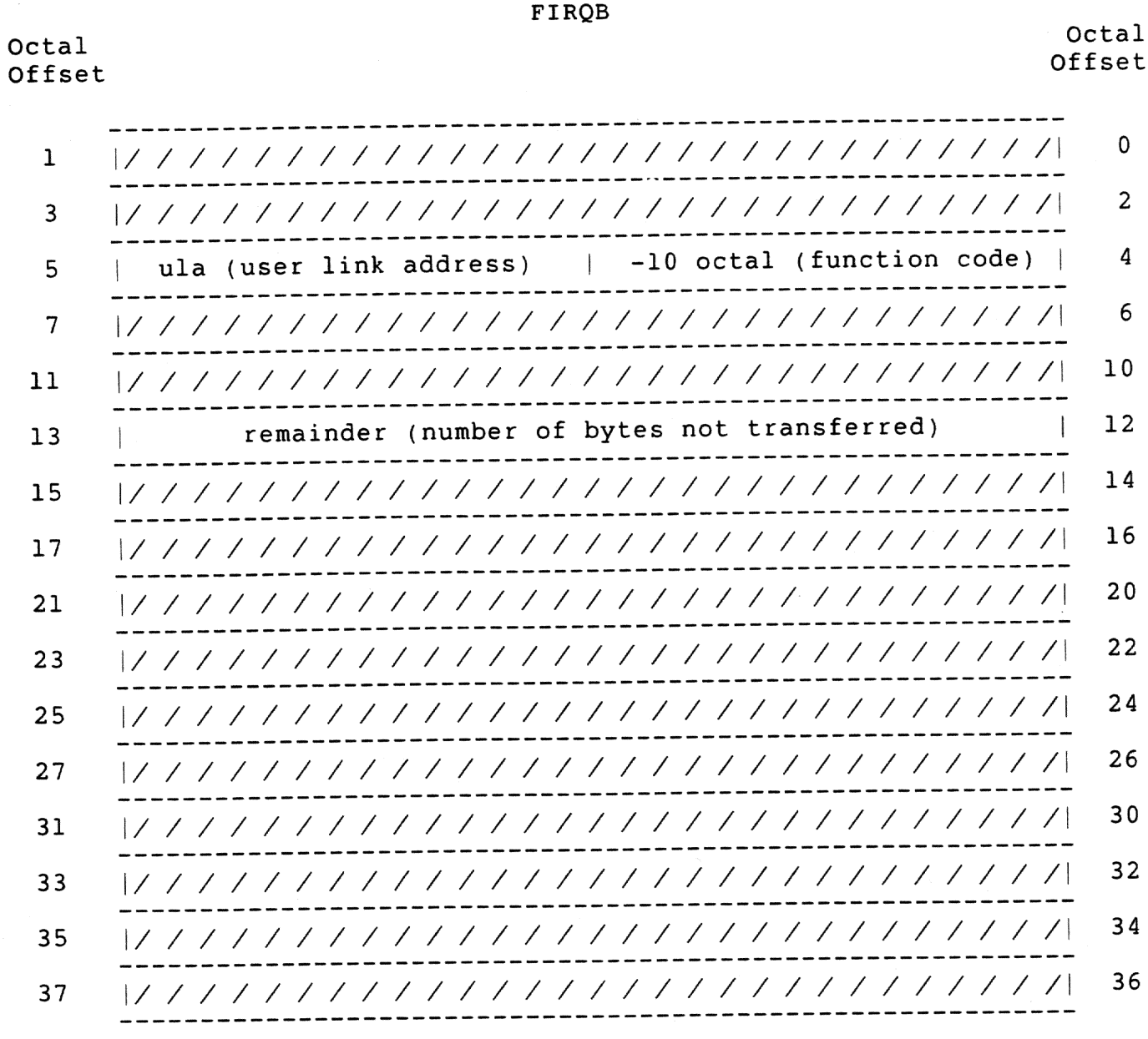


XRB+2 — Always zero for Link Service Messages. (This word is the length field in other returned messages.)

Format of Received Disconnect Message

A received Disconnect Message indicates that an established logical link with a remote program has been terminated by the remote program. All other messages sent by the remote program over this logical link have been received by this program. The following format describes the data returned to the FIRQB and XRB. Up to 16. bytes of user message data can be returned to the buffer specified in the Receive call.

FIRQB



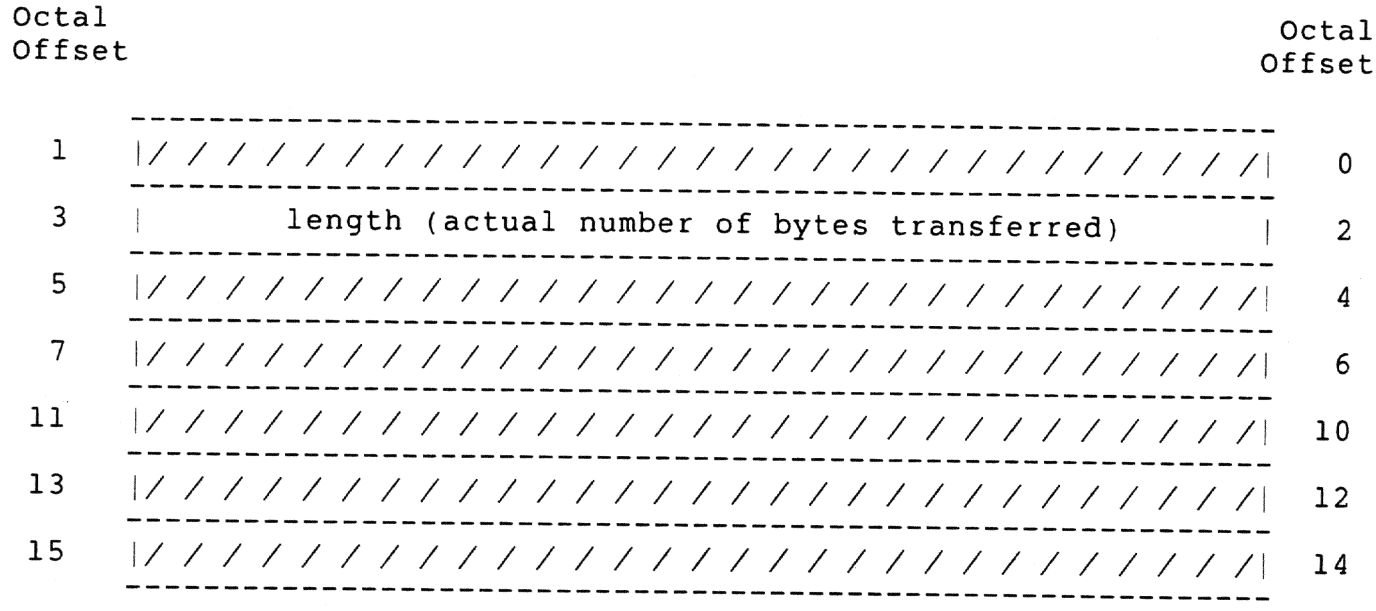
FIRQB+4 Function code for a Disconnect Message.

Format of Received Disconnect Message
MACRO

FIRQB+5 — User Link Address, previously defined in this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 — Number of bytes of user message data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate flag was set in the *rmod* argument for the Receive call.

XRB

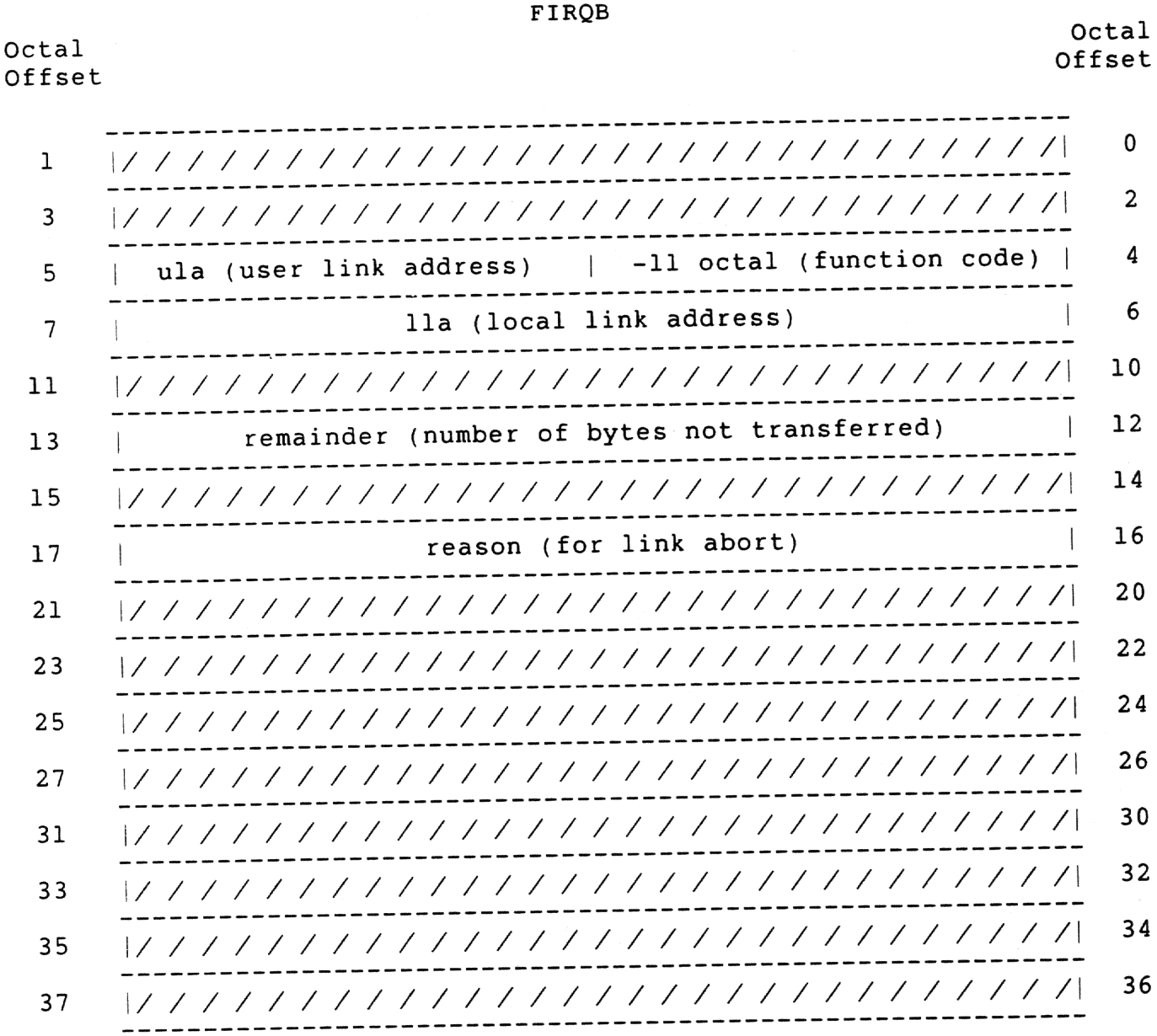


XRB+2 — The length (bytes) of information transferred to the buffer on this Receive call.

Format of Received Link Abort Message

A received Link Abort Message indicates that a logical link has been terminated by the remote program or by the local or remote ECL. The following format describes the data returned to the FIRQB and XRB. Up to 16. bytes of user message data can be returned to the buffer specified in the Receive call.

FIRQB



FIRQB+4 Function code for Link Abort Message.

Format of Received Link Abort Message
MACRO

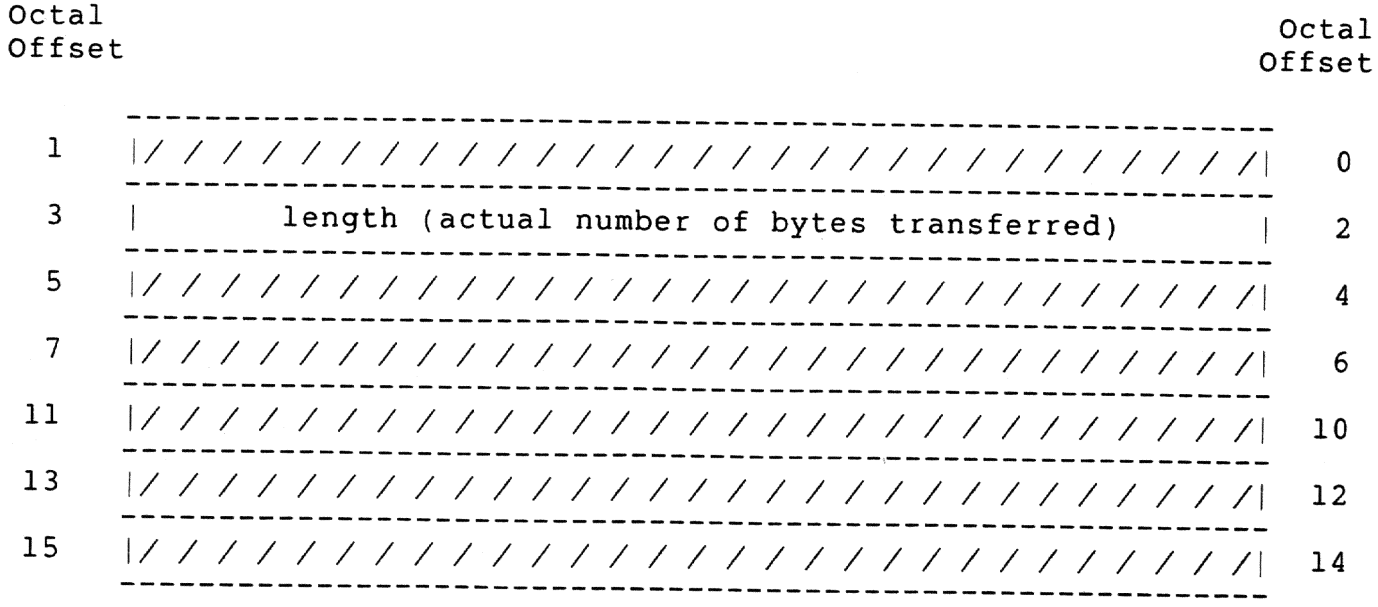
FIRQB+5 — User Link Address, if any, previously defined in the receiving program's Connect Initiate or Connect Confirm Message.

FIRQB+6 — Local Link Address established by local ECL for this link. Identifies the link if the remote program sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No ULA exists under these conditions to identify the link being aborted.

FIRQB+12 — Number of bytes of user data not transferred to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate flag was set in the *rmod* argument for the Receive call.

FIRQB+16 — Abort reason. If this value is nonzero, the link was aborted by either the local or remote ECL and the reason is specified in this word. The reasons that apply for a link abort are listed in Appendix B. If this word is zero, the link was aborted at the request of the remote program.

XRB



XRB+2 — The length (bytes) of information transferred to the buffer on this Receive call.

Chapter 12

FIRQB and XRB Layouts for MACRO Message Calls

When one of the MACRO message calls described in Chapter 11 is used in a MACRO program, subroutine $$$MSR is invoked. This subroutine first sets up the job's File Request Queue Block (FIRQB) and Transmit Request Block (XRB), using parameters passed to it by the macro call. The FIRQB and XRB are two areas within the first 512. bytes of virtual address space reserved for communications between the RSTS/E monitor, the run-time system, and the user job. A general description of the first 512. bytes of virtual memory, along with the FIRQB and the XRB, can be found in the *RSTS/E System Directives Manual*. It then executes a .MESAG monitor directive. This directive transfers program control to the appropriate section of the RSTS/E monitor, which then performs the function requested.

This chapter contains the FIRQB and XRB layouts for each of the message calls. These layouts are provided for reference purposes only. Do not attempt to set up the FIRQB and the XRB and invoke the .MESAG directive yourself. Use only the MACRO calls (see Chapter 11) to access the DECnet/E Send/Receive operations.

This chapter does not give a detailed description of the parameters passed. However, these parameters do carry the same mnemonic abbreviations as their corresponding MACRO parameters in Chapter 11. See Chapter 11 for a detailed explanation of these parameters.

Declare Receive

FIRQB

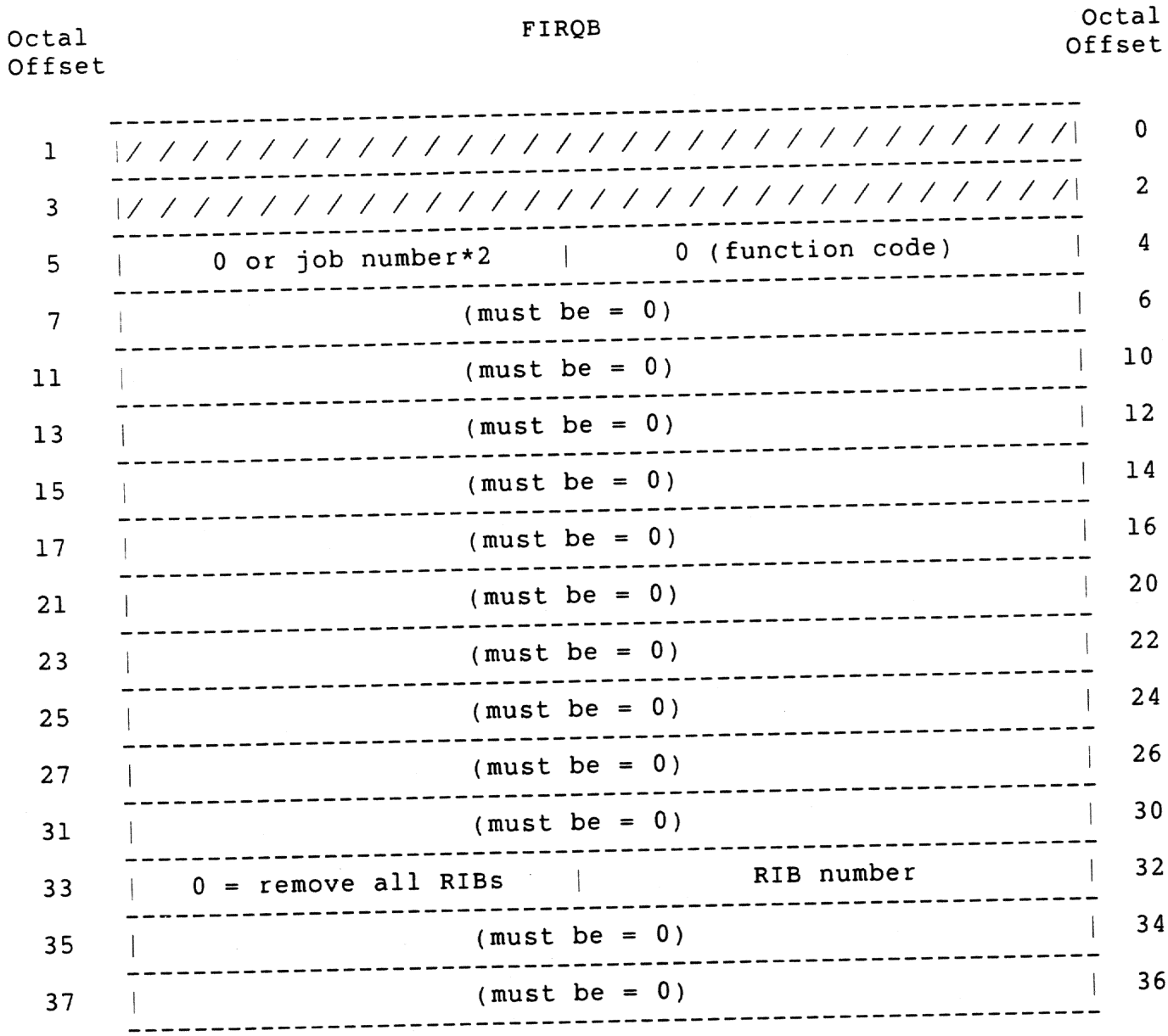
Octal Offset		Octal Offset
1	//	0
3	//	2
5	//////////////// | 1 (function code)	4
7	name	6
11	(receiver logical name in ASCII, space fill to six bytes)	10
13		12
15	access code | objtyp (object type)	14
17	//	16
21	lmax (link maximum) | mmax (message maximum)	20
23	packet maximum	22
25	packets/message | outbound link maximum	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0) | RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Declare Receiver subfunction of .MESAG.

Note that you need SYSIO privilege to declare a receiver with an unrestricted name, network server, or local object. You need EXQTA privilege to suppress RIB or message quota checks.

Remove Receiver



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Remove Receiver subfunction of .MESAG.

Note that you need JOBCTL privilege to remove the RIB of another job.

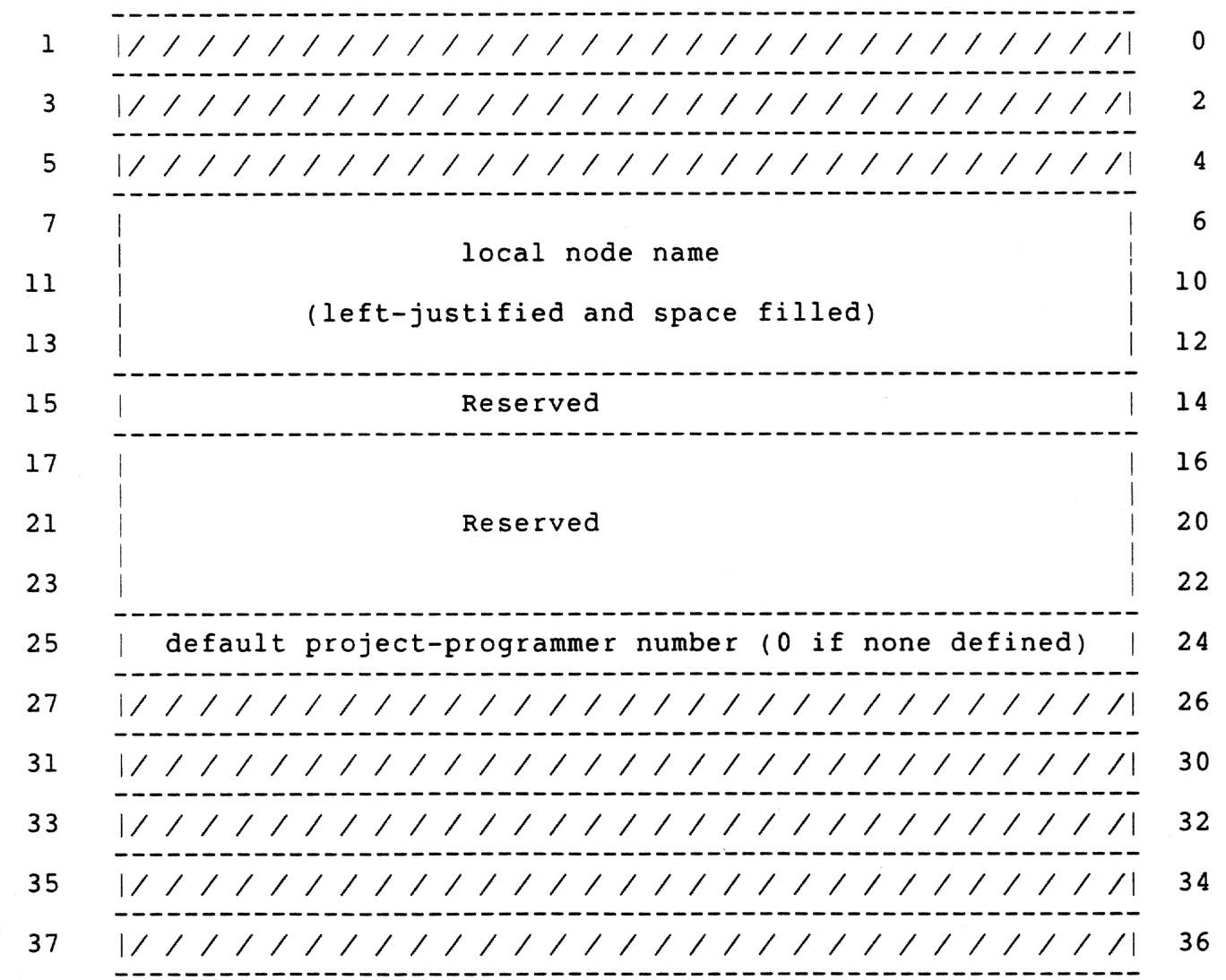
Get Local Node Parameters

Octal Offset	FIRQB	Octal Offset
1	//////////////////////////////////	0
3	//////////////////////////////////	2
5	1 | -19 (function code)	4
7	(must be = 0)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	14
17	(must be = 0)	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	32
35	(must be = 0)	34
37	(must be = 0)	36

Data Returned

In addition to a status code in byte 0, the following data is returned in the FIRQB.

FIRQB


Octal Offset
Octal Offset
1
0
3
2
5
4
7
6
local node name
11
10
(left-justified and space filled)
13
12
15
Reserved
14
17
16
21
Reserved
20
23
22
25
default project-programmer number (0 if none defined)
24
27
26
31
30
33
32
35
34
37
36

Log User Event

Octal Offset	FIRQB	Octal Offset
1	//////////////////////////////////	0
3	//////////////////////////////////	2
5	evmod (event modifier) | -10 (function code)	4
7	event code (bits 0-4 = evtyp; bits 6-14 = evcls)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	enval (entity value or 0 if no entity)	14
17	(must be = 0) | entyp (entity type)	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	32
35	(must be = 0)	34
37	(must be = 0)	36

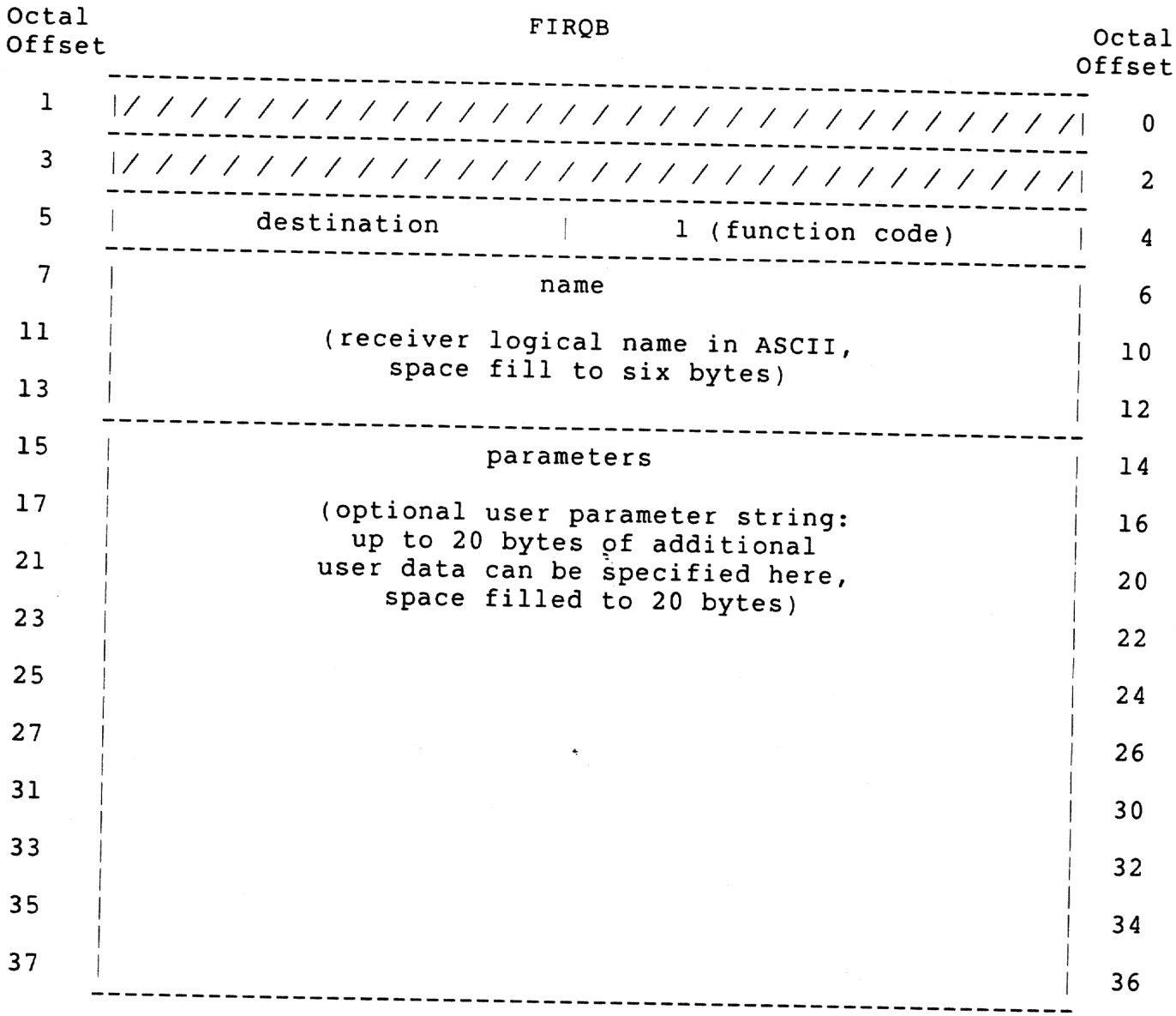
XRB

Octal Offset		Octal Offset
1	length of output buffer, in bytes (0 - 200)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned for the Log User Event subfunction of .MESAG.

Send Local Data Message



Note that you need SEND privilege to send to a restricted receiver.

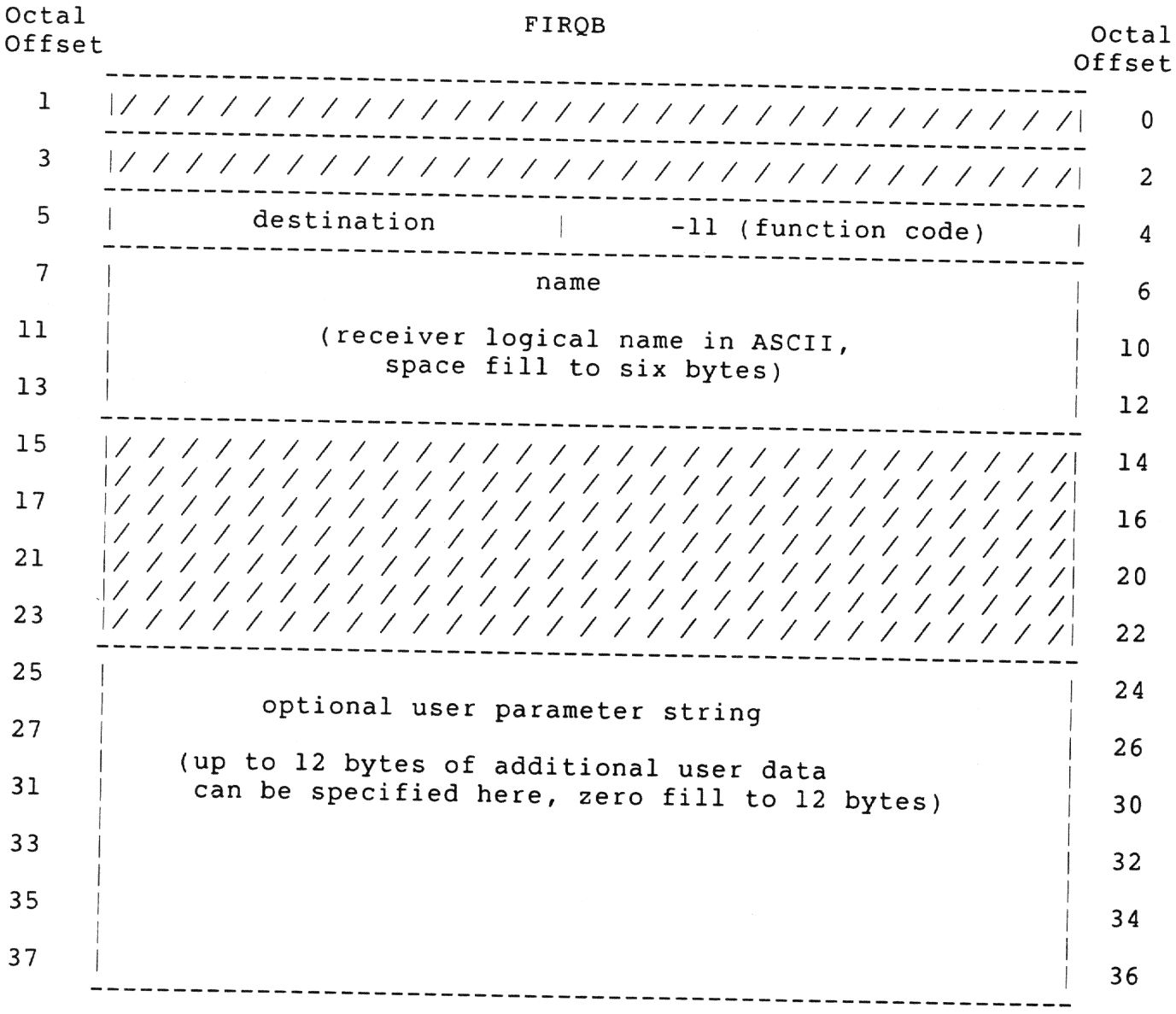
XRB

Octal Offset		Octal Offset
1	length of output buffer, in bytes (0 - 512)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

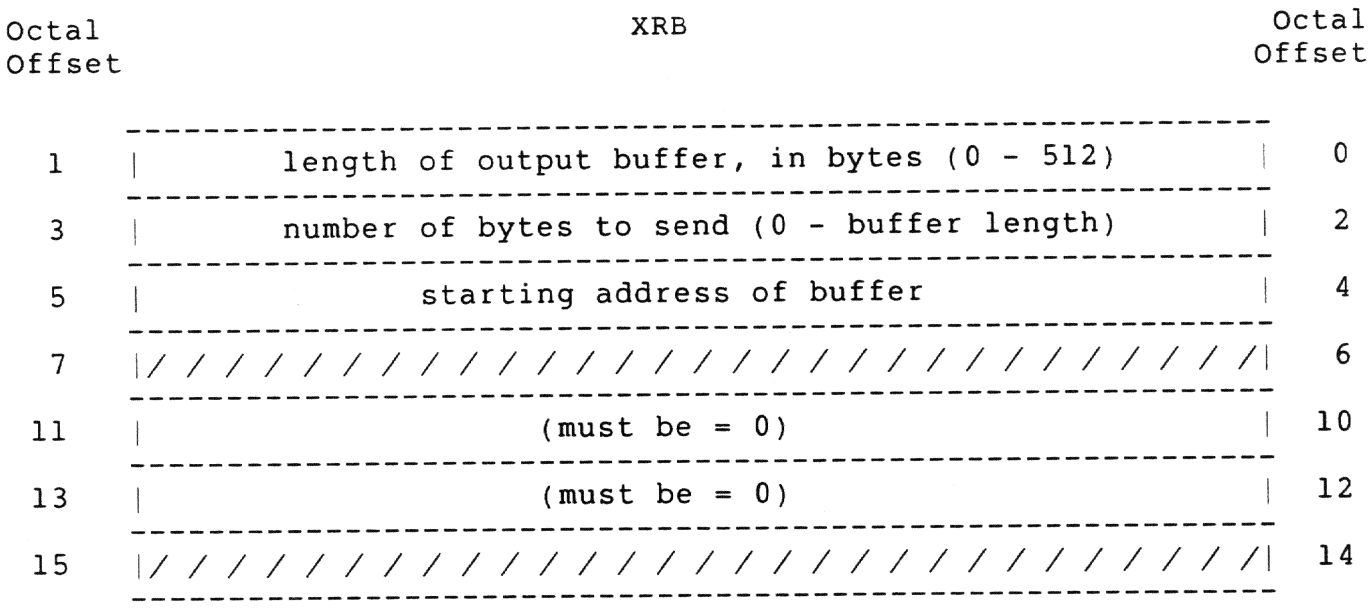
Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Local Data Message subfunction of .MESAG.

Send Local Data Message with Privilege Mask



Note that you need SEND privilege to send to a restricted receiver.



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Local Data Message with Privilege Mask subfunction of .MESAG.

Send Connect Initiate Message

Octal Offset	FIRQB	Octal Offset
1	//////////////////////////////////	0
3	//////////////////////////////////	2
5	ula (user link address) | 2 (function code)	4
7	(must be = 0)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	14
17	(must be = 0) | llmod (link modifier)	16
21	rmax (maximum bytes in received network data message)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0) | RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

XRB

Octal Offset		Octal Offset
1	length of output buffer, in bytes (172 - 188)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned for the Send Connect Initiate subfunction of .MESAG.

Send Connect Confirm Message

FIRQB

Octal Offset		Octal Offset
1	////////////////////////////////|	0
3	////////////////////////////////|	2
5	ula (user link address)	3 (function code)	4
7	lla (local link address)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	14
17	(must be = 0)	llmod (link modifier)	16
21	rmax (maximum bytes in received network data message)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

Octal Offset	XRB	Octal Offset
1	length of output buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

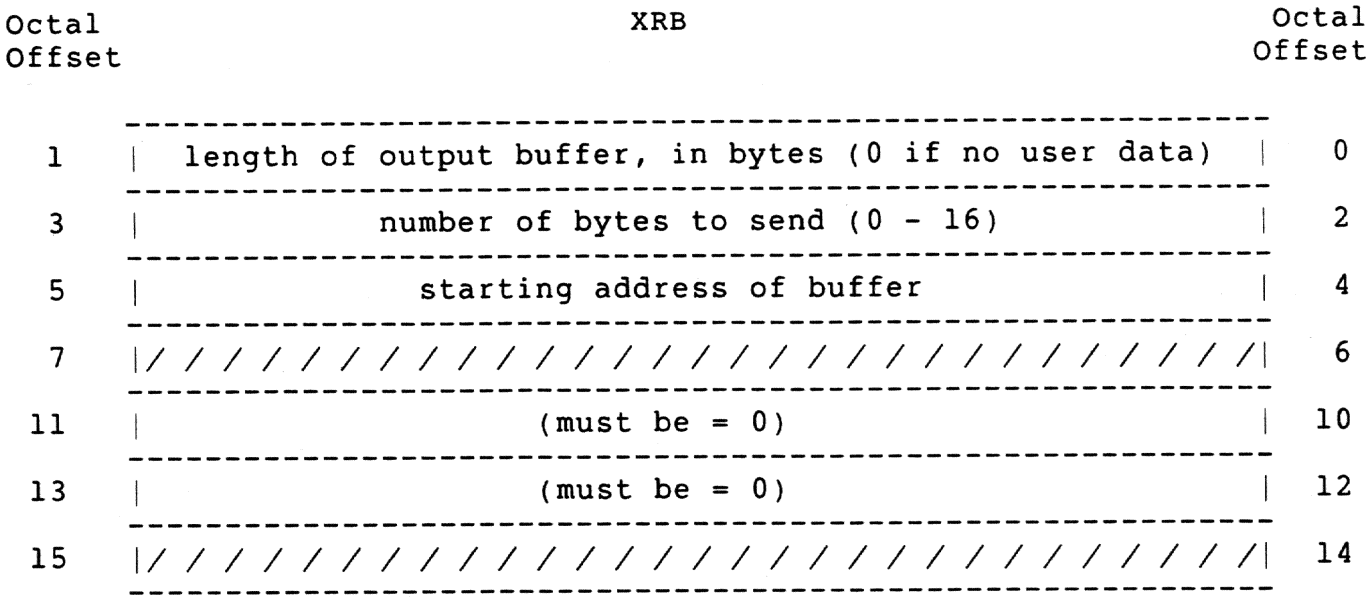
Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Connect Confirm subfunction of .MESAG.

Send Connect Reject Message

FIRQB

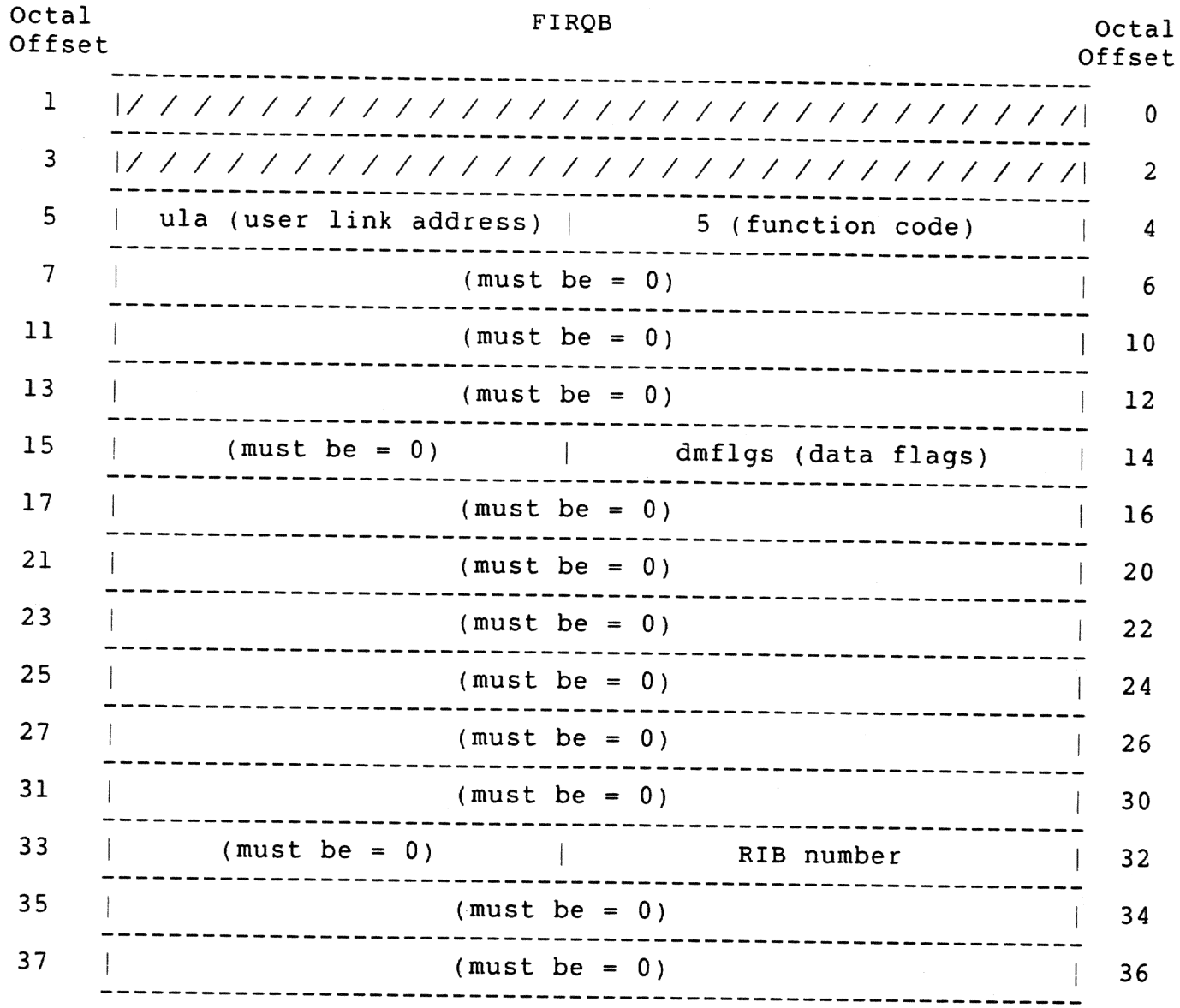
Octal Offset		Octal Offset
1	/ /	0
3	/ /	2
5	/ / / / / / / / / / / / / / / / | 4 (function code)	4
7	lla (local link address)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	14
17	reason code	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0) | RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Connect Reject subfunction of .MESAG.

Send Network Data Message



Octal Offset	XRB	Octal Offset
1	length of buffer, in bytes	0
3	number of bytes to send (0 - 532)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned

If bit 7 is set in the *dmflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message.

Send Interrupt Message

FIRQB

Octal Offset		Octal Offset
1	/ /	0
3	/ /	2
5	ula (user link address) | 6 (function code)	4
7	(must be = 0)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0) | inflgs (interrupt flags)	14
17	(must be = 0)	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0) | RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

Octal Offset	XRB	Octal Offset
1	length of buffer, in bytes	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

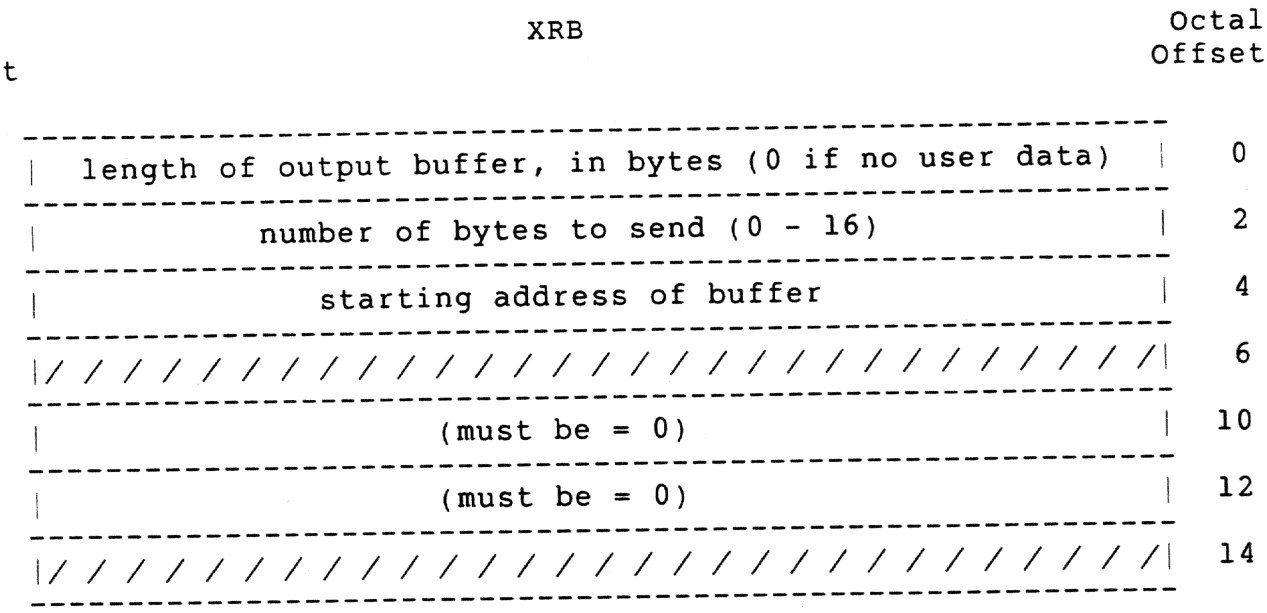
Data Returned

If bit 7 is set in the *inflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message.

Send Link Service Message

FIRQB

Octal Offset		Octal Offset
1	////////////////////////////////|	0
3	////////////////////////////////|	2
5	ula (user link address)	7 (function code)	4
7	(must be = 0)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	lsflgs (link service flags)	14
17	ircnt (interrupt req cnt)	drcnt (data req count)	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36



Data Returned

If bit 7 is set in the *lsflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message.

Send Disconnect Message

FIRQB

Octal Offset		Octal Offset
1	////////////////////////////////|	0
3	////////////////////////////////|	2
5	ula (user link address)	8 (function code)	4
7	////////////////////////////////|	6
11	////////////////////////////////|	10
13	////////////////////////////////|	12
15	////////////////////////////////|	14
17	////////////////////////////////|	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

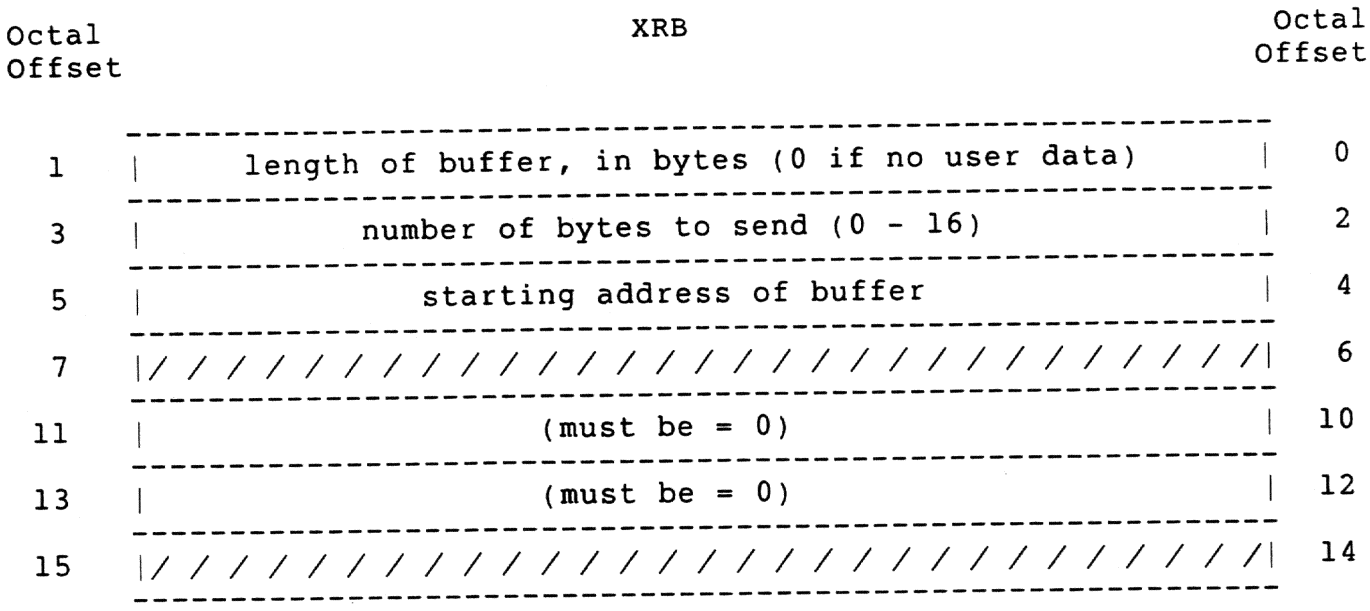
Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Disconnect subfunction of .MESAG.

Send Link Abort Message

FIRQB

Octal Offset		Octal Offset
1	////////////////////////////////|	0
3	////////////////////////////////|	2
5	ula (user link address)	9 (function code)	4
7	(must be = 0)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	(must be = 0)	14
17	(must be = 0)	16
21	(must be = 0)	20
23	(must be = 0)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0)	RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Link Abort subfunction of .MESAG.

Receive Message

Octal Offset	FIRQB	Octal Offset
1	//////////////////////////////////	0
3	//////////////////////////////////	2
5	rmod (receive modifiers) | 2 (function code)	4
7	qual (normally 0) | sndr (sender select)	6
11	(must be = 0)	10
13	(must be = 0)	12
15	//////////////////////////////////	14
17	//////////////////////////////////	16
21	//////////////////////////////////	20
23	slptime (sleep time, in seconds)	22
25	(must be = 0)	24
27	(must be = 0)	26
31	(must be = 0)	30
33	(must be = 0) | RIB number	32
35	(must be = 0)	34
37	(must be = 0)	36

Octal Offset	XRB	Octal Offset
1	length of buffer, in bytes (0 if no data desired)	0
3	(must be = 0)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned

The Receive call returns data identifying the type of message received to the FIRQB and XRB, and user data (if any) to the buffer defined at XRB+4.

The FIRQB and XRB formats for the various message types are shown in Chapter 11, Sections "Format of Received Local Data Message" through "Format of Received Link Abort Message."

Appendix A

Network Object Type Codes

Network object type codes simplify the identification of programs performing identical functions at different nodes of a network. Table A-1 lists the network object type codes currently assigned for DECnet/E use.

Table A-1: Network Object Type Codes

Code	Type of Process
000	General Task, User Process
001	File Access (FAL/DAP Version 1)
002	Unit Record Services (URD)
003	Application Terminal Services (ATS)
004	Command Terminal Services (CTS)
005	RSX-11M Task Control, Version 1
006	Operator Services Interface
007	Node Resource Manager
008	IBM 3270-BSC Gateway
009	IBM 2780-BSC Gateway
010	IBM 3790-SDLC Gateway
011	TPS Application
012	RT-11 DIBOL Application

Table A-1: Object Type Codes (Cont.)

Code	Type of Process
013	TOPS-20 Terminal Handler
014	TOPS-20 Remote Spooler
015	RSX-11M Task Control, Version 2
016	TLK Utility (LSN on DECnet/E)
017	File Access (FAL/DAP Version 4 and later)
018	RSX-11S Remote Task Loader
019	Network Management Listener (NICE Process)
020	RSTS/E Media Transfer Program (NETCPY)
021	Reserved for DIGITAL use
022	Message Router
023	Host Terminal Handler (NPKDVR)
024	Concentrator Terminal Handler
025	Loopback Mirror
026	Event Receiver
027	VAX/VMS Personal Message Utility
028	File Transfer Spooler (FTS)
029-062	Reserved for DIGITAL use
063	DECnet test tool (DTR)
064-127	Reserved for DIGITAL use
128-255	Reserved for customer extensions

Appendix B

Reason Codes for Connect Reject and Link Abort

When a remote program request for a logical link is rejected or a logical link is aborted, a code is returned identifying the reason for such action. Table B-1 lists these rejection codes.

Table B-1: Reason Codes

Code	Reason
000	No error -- user-initiated reject or abort
001	Resource allocation failure
002	Destination node does not exist
003	Node shutting down
004	Destination program does not exist
005	Invalid destination name or source name field
006	Destination program's queue full
007	Unspecified error condition
008	Third party aborted logical link
009	User-initiated link abort
010-020	Reserved
021	Invalid destination address in Connect Initiate Message
022	Invalid destination address in Connect Confirm Message

Table B-1: Reason Codes (Cont.)

Code	Reason
023	Source address zero in Connect Initiate or Connect Confirm Message
024	Flow control violation -- invalid request count in Link Service Message
025-031	Reserved
032	Too many connects to node
033	Too many connects to destination program
034	Access not permitted
035	Logical link services mismatch
036	Invalid accounting information
037	Segment size too small
038	User aborted, timed out, or canceled link
039	Node is unreachable
040	Flow control failure -- data received when request count zero
041	No current link (cannot recognize destination address)
042	Confirmation from remote system of Disconnect Message
043	Image data field too long
044-049	Reserved
050-056	The remote system requires security features that DECnet does not implement

-A-

Access
  consistency with receiver identity, 5-10, 6-8, 9-10, 10-8, 11-12
  specifying, 5-8, 6-6, 9-8, 10-6, 11-10
  types permitted, 5-8, 6-7, 9-9, 10-7, 11-11
Access control fields, 10-36, 10-87, 11-49
Accounting information, 5-58, 6-49, 9-51, 10-36, 10-49, 10-87, 11-49, 11-62
Area routing, 1-4
Arguments
  bit values, 9-4, 9-5
  byte values, 9-2
  in system calls, 5-3, 6-2
  input, 8-2, 9-2, 10-2
  output, 9-2, 10-2
Assembling, 11-6
  library files, 11-6
  MAC assembler, 11-6
  MACRO assembler, 11-6
Automatic start-up, 4-3, 4-4, 4-5
  ordering of objects, 4-6
  starting line number, 4-6
  timing, 4-7

-B-

Backpressure flow control, 4-11, 8-11
  clearing, 4-11
  effect on DECnet/E programs, 4-11
  use of Link Service Message, 4-11
Buffer
  allocating, 5-2, 6-2, 10-2
  assigning for messages, 5-97
  declared maximum, 9-11, 10-9, 11-14
  limited space, 5-38, 11-43
  monitor pool, 2-10, 3-3, 3-7
Buffer arguments, 6-4

-C-

CALL statement, 9-1
Channel number, 5-2
Codes, network object type, A-1
COMMON statement, 6-2
COMMON.MAC, 11-6
Comparison of interfaces, 7-5
Compilers, 10-4
Compiling, 8-2, 9-6, 10-4
Concise interface, 7-1
  accounting information, 8-5, 8-8
  comparison with extended interface, 7-5
  Connect Accept (CNTACP), 7-2, 8-7
  Connect Request (CNTCON), 7-1, 8-3
  deadlock situations, 7-2
  Disconnect (CNTDIS), 7-2, 8-17
  files, 7-1
  form of calls, 8-1
  password, 8-4, 8-8
  PPN, 8-4
  Receive Record (CNTRCV), 7-2, 8-14
  receiver identity, 8-3, 8-7, 8-8
  records, 7-1
  Send Record (CNTSND), 7-2, 8-11
  synchronous I/O, 7-2
  UIC, 8-8
Connect Confirm Message, 3-11, 7-4
  received format, 5-109, 6-94, 9-97, 10-93, 11-112
  send call, 5-50, 6-44, 9-46, 10-44, 11-54
Connect Data Block, 3-9, 3-10, 5-39, 5-40, 5-103, 11-43
Connect Initiate Message, 3-9, 7-4
  local program identification, 10-36, 10-87, 11-49

Connect Initiate Message (Cont.)
received format, 5-101, 6-86, 9-89, 10-85, 11-103
remote program identification, 10-36, 10-87, 11-49
send call, 5-36, 6-32, 9-34, 10-32, 11-41
Connect Reject Message, 3-12, 7-4
received format, 5-111, 6-96, 9-99, 10-95, 11-115
send call, 5-56, 6-49, 9-51, 10-49, 11-60
CORCMN, 4-6
Core common, 4-6
CTRL/C, 3-7

-D-

DAP, 1-5
Data Division, 8-2, 9-2
Data transmit queue, 4-8, 5-80, 5-85, 5-117, 6-102, 9-104, 10-99, 11-122
DDCMP, 1-4
Deadlock situations, 7-2
Declare Receiver, 3-3, 3-4, 7-3
access, 5-8, 6-6, 9-8, 10-6, 11-10
declaring intended usage, 3-6
link maximum, 3-6
message maximum, 3-6
sleep bit, 5-9, 6-7, 9-10, 10-8, 11-12
SYSIO privilege, 10-7
system call, 5-4, 6-5, 9-7, 10-5, 11-8
DECnet/E, 9-2
Default project-programmer number, 3-7, 5-18, 6-15, 9-17, 10-16, 11-20
DEFINE OBJECT, 4-6
Device
I/O channel, 5-2
null device, 5-2
opening, 5-2
DIM statement, 6-2
Disconnect Message, 3-14, 7-5
affected by transmit queues, 4-9
received format, 5-119, 6-104, 9-106, 10-103, 11-125
Disconnect Message (Cont.)
send call, 5-80, 6-69, 9-72, 10-68, 11-82
DNA, 1-3, 5-20, 6-18, 11-24

-E-

ECL, 2-1, 6-7, 8-2
End node, 1-5
EQUIVALENCE statement, 10-3
ERRCPY, 6-79, 9-81, 10-78, 11-94
Errors
CNTACP (Concise interface), 8-9
CNTCON (Concise interface), 8-5
CNTDIS (Concise interface), 8-17
CNTRCV (Concise interface), 8-15
CNTSND (Concise interface), 8-12
Declare Receiver, 5-13, 6-11, 9-13, 10-12, 11-15
Get Local Node Parameters, 5-19, 6-16, 9-18, 10-17, 11-22
Log User Event, 5-23, 6-20, 9-22, 10-21, 11-27
Receive, 5-97, 6-82, 9-85, 10-82, 11-97
Remove Receiver, 5-17, 6-14, 9-16, 10-15, 11-19
Send Connect Confirm Message, 5-53, 6-47, 9-49, 10-47, 11-57
Send Connect Initiate Message, 5-46, 6-41, 9-43, 11-50
Send Connect Reject Message, 5-59, 6-51, 9-53, 10-51, 11-63
Send Disconnect Message, 5-82, 6-71, 9-74, 10-70, 11-84
Send Interrupt Message, 5-71, 6-61, 9-64, 10-61, 11-74
Send Link Abort Message, 5-87, 6-75, 9-77, 10-74, 11-89
Send Link Service Message, 5-77, 6-66, 9-69, 10-65, 11-79
Send Local Data Message, 5-30, 6-26, 9-28, 10-26, 11-34, 11-40

Errors (Cont.)
Send Local Data Message with Privilege Mask, 5-35, 6-31, 9-33, 10-31
Send Network Data Message, 5-65, 6-56, 9-58, 10-56, 11-68
Ethernet, 1-4
Extended interface, 7-3
comparison with concise interface, 7-5

-F-

F77 compiler (FORTRAN-77), 10-4
FIP, 5-1
FIRQB, 11-6, 11-91
input to .MESAG, 12-1
Flow control, 2-10, 3-13, 4-7
local congestion, 4-19
local modifier, 5-38, 6-32, 9-34, 9-47, 10-32, 10-44, 11-42, 11-56
programming hints, 4-19
reasons for inhibiting transmission, 4-19
remote congestion, 4-19
remote modifier, 5-101, 5-109, 6-86, 6-94, 9-97, 10-93, 11-103
use of Link Service Message, 4-19
FORTRAN compiler, 10-4
FORTRAN-77 compiler (F77), 10-4
Full-duplex transmission, 2-3

-G-

Get Local Node Parameters
system call, 5-18, 6-15, 9-17, 10-16, 11-20

-H-

Hollerith constants, 10-2
Hop, 1-1

-I-

I/O channels, 5-2
INTEGER*2 variables, 10-2
INTEGER*4 variables, 10-2
Interrupt Message, 2-9, 3-13, 7-5
queueing, 2-9, 3-13, 5-68, 5-114, 6-59, 9-62, 9-101, 10-59, 10-98, 11-71, 11-119
received format, 5-114, 6-99, 9-101, 10-98, 11-119
reenabling, 3-13, 3-14, 4-17
request count, 4-17, 5-76, 5-117, 6-66, 6-102, 9-69, 9-104, 10-65, 10-99, 11-78, 11-122
send call, 5-68, 6-59, 9-62, 10-59, 11-71
Interrupt Message flow control, 3-13, 4-17
effect on DECnet/E receivers, 4-17
effect on DECnet/E senders, 4-17
use of Link Service Message, 4-17
Interrupt/Link Service queue, 4-8, 5-80, 5-85, 5-117, 6-102, 9-104, 10-99, 11-122

-J-

Job
in receiver sleep, 5-95
number in Receive, 6-78, 9-80, 10-78, 11-93
number of local receiver, 3-8, 5-27, 6-23, 6-28, 10-23, 10-28, 11-31
privileges, 5-3
Receiver ID Block, 5-4, 6-5, 9-7, 10-5, 11-8
receiving from system job 0, 6-79, 9-81, 10-78, 11-94
Remove Receiver, 3-7, 5-16, 6-13, 9-15, 10-14, 11-18
RIB, 3-6

-L-

LINK, 10-4, 11-7
Link
physical, 1-1, 2-1
Link Abort Message, 3-15, 7-5
received format, 5-120, 6-105, 9-107, 10-104, 11-127

Link Abort Message (Cont.)
send call, 5-85, 6-73, 9-76, 10-72, 11-87
unaffected by transmit queues, 4-9
Link master, 2-2
Link Service Message, 3-13, 3-14, 7-5
purpose, 3-14, 4-11, 5-75, 11-77
received format, 5-115, 6-100, 9-102, 10-99, 11-121
send call, 5-74, 6-64, 9-67, 10-63, 11-76
Link status bit, 5-64, 10-54, 11-67
Link status flags
local, 5-115, 6-100, 9-102, 10-99, 11-122
remote, 5-116, 6-101, 9-103, 10-99, 11-122
Link status information, 3-13, 3-14, 5-115, 9-102, 10-3, 10-99, 11-121
Linking, 8-2, 9-6, 10-4, 11-7
LINK, 10-4, 11-7
TKB, 10-4, 11-7
LLA, 2-7, 3-10, 3-11, 3-12, 3-15, 5-51, 5-57, 6-44, 6-49
Local communication, 3-1, 3-4, 3-8, 5-26, 5-32, 6-23, 6-28
Local Data Message, 3-8, 7-4
received format, 5-99, 6-84, 9-87, 10-83, 11-99
send call, 5-26, 6-23, 9-25, 10-23, 11-30
Local Data Message with Privilege Mask, 7-4
received format, 5-100, 6-85, 9-88, 10-84, 11-101
send call, 5-32, 6-28, 9-30, 10-28, 11-36
Local link status flags, 5-115, 6-100, 9-102, 10-99, 11-122
Local node, 3-7
Local node parameters, 3-3, 3-7, 7-4
Local program, 3-6
identification, 10-36, 10-87, 11-49
Log User Event
system call, 5-20, 6-18, 9-20, 10-19, 11-24
Log-in information, 10-36, 10-87, 11-49
Logical full-duplex transmission, 2-3
Logical link
active, 3-7
creation, 2-2
declared maximum, 3-6, 4-3, 4-4, 5-8, 5-9, 5-11, 10-7, 11-11
declared message maximum, 3-6, 4-4, 5-11, 5-118, 6-103, 9-105, 10-99, 11-122
DECnet/E limitations, 2-11
efficient line usage, 2-6
identifiers, 2-2, 2-7
message count, 5-118, 6-103, 9-105, 10-99, 11-122
message streams, 2-8, 3-13
Logical message, 4-13, 5-63, 9-100, 10-53, 11-67
end of message segment, 4-14
request count, 4-14, 5-76, 5-116, 5-117, 6-101, 6-102, 9-103, 9-104, 10-99, 11-122
LOGICAL*1 variables, 10-2, 10-3

-M-

MAC assembler, 11-6
MACRO assembler, 11-6
MACRO subroutines, 1-5
MACRO-11 subprograms, 7-3, 8-2, 9-1, 9-2
MACRO-11 subroutines, 4-6, 4-7, 10-1
MAP statement, 6-2, 6-65
Master-slave communication, 3-15, 5-80, 6-69, 9-72, 10-68, 11-82
MCALL directive, 11-1
MESAG monitor directive, 11-2, 12-1
Message
assigning buffer, 5-97
count for logical link, 5-118, 6-103, 9-105, 10-99, 11-122
count of pending messages, 9-105, 10-99, 11-122

Message (Cont.)
declared maximum, 3-6, 4-3, 4-4, 5-11, 5-118, 6-103, 9-105, 10-99, 11-14
limit of transmit queues, 4-9
request count, 4-14
Message flow control, 4-13
Message streams, 2-7, 2-8, 3-13
Messages
count of pending messages, 5-118, 6-103
MOVE statement, 8-2, 9-2
MSR macro, 11-1
Multiple copies of objects, 3-5, 4-2, 4-4
under PBS, 4-5
via automatic start-up, 4-5
via terminal users, 4-4

-N-

NCP, 1-6, 4-5, 4-8
NET, 1-5
NETCPY, 1-6
Network, 1-1, 1-3
as a file, 7-1
Network addressing, 3-5, 4-1
handling incoming requests, 4-2
permitted by DECnet/E, 4-3
to name, 4-4
to object type code, 4-3
Network Data Message, 3-12, 7-4
received format, 5-112, 6-97, 9-100, 10-96, 11-117
send call, 5-61, 6-53, 9-55, 10-53, 11-65
Network diameter, 1-1
Network Information and Control Exchange Protocol
See NICE
Network object type codes, A-1
Network program, 3-4, 3-6
NFT/FAL, 1-5, 5-9, 6-7, 9-10, 10-8, 11-12
NICE, 3-8, 5-20, 6-18, 6-19, 10-19, 11-24
Node, 1-1
address, 3-7, 5-18, 6-15, 9-17, 10-16, 11-20
alias, 9-17, 11-20
local, 3-7
Node (Cont.)
name, 3-7, 5-18, 6-15, 8-3, 9-17, 10-16
nonrouting, 1-5
number, 3-7, 5-18, 6-15, 10-16
routing, 1-5
Nonrouting node, 1-5
NSP, 1-4

-O-

Object type code
receiver identity, 8-4
One-shot bit, 5-9, 6-7, 9-10, 10-7, 11-12
Opening devices, 5-2

-P-

Parameters
with DEFINE or SET OBJECT, 4-6
Password, 5-58, 6-49, 9-51, 10-36, 10-49, 10-87, 11-49
Path, 1-1
Path length, 1-1
Pending message queue, 2-9, 3-4, 3-7, 3-14, 3-15, 4-11, 5-68, 5-118, 6-103
Phase III, 1-4
Phase IV, 1-4
Physical link, 1-1, 2-1
PICTURE statement, 8-2, 9-2
PPN, 5-58, 6-49, 9-51, 11-49, 11-61
default, 3-7, 5-18, 6-15, 9-17, 10-16, 11-20
Privileges
job, 5-3
Procedure Division, 8-2, 9-2
Program
local, 3-6
network, 3-4, 3-6
remote, 1-2
source, 2-2
target, 2-2
Program-regulated flow control, 2-11, 4-12
selection, 3-9, 3-11, 4-13, 5-38, 5-101, 5-109, 6-86, 6-94, 9-97, 10-32

Program-regulated flow control (Cont.)
use of Link Service Message, 4-13
Protocols, 1-3
DAP, 1-5
DDCMP, 1-4
Ethernet, 1-4
NICE, 3-8, 5-20, 6-18, 6-19, 9-20, 10-19, 11-24, 11-26
NSP, 1-4
Routing, 1-4

-Q-

Queue
Data transmit, 4-8, 5-117, 6-102, 9-104, 10-99 11-122
Interrupt/Link Service, 4-8, 5-117, 6-102, 9-104, 10-99, 11-122
management, 4-8
message limit, 4-9
pending message, 2-9, 3-4, 3-7, 3-14, 3-15, 4-11, 5-68, 5-118, 6-103

-R-

Reason code, A-2
for Connect Reject, 3-12, 5-58, 5-111, 6-49, 6-96, 10-49, 11-116
for Link Abort, 3-16, 5-120, 6-105, 9-107, 10-104, 11-127
Receive, 3-4, 7-5
selective, 3-4, 3-14, 3-15, 6-78
sleep, 3-4, 6-78, 6-81, 11-92, 11-96
system call, 5-90, 6-77, 9-79, 10-76, 11-91
truncation of message, 3-4, 6-78, 11-92
Receive maximum, 3-9, 3-10, 3-11, 3-13, 5-38, 5-101, 5-109, 6-33, 6-45, 6-86, 6-94, 9-35, 9-89, 9-97, 10-33, 10-45, 10-93, 11-43
Received formats (BASIC-PLUS)
Connect Confirm Message, 5-109
Connect Initiate Message, 5-101
Connect Reject Message, 5-111
Disconnect Message, 5-119
Interrupt Message, 5-114
Link Abort Message, 5-120
Link Service Message, 5-115
Local Data Message, 5-99
Local Data Message with Privilege Mask, 5-100
Network Data Message, 5-112
Received formats (BASIC-PLUS-2)
Connect Confirm Message, 6-94
Connect Initiate Message, 6-86
Connect Reject Message, 6-96
Disconnect Message, 6-104
Interrupt Message, 6-99
Link Abort Message, 6-105
Link Service Message, 6-100
Local Data Message, 6-84
Local Data Message with Privilege Mask, 6-85
Network Data Message, 6-97
Received formats (COBOL)
Connect Confirm Message, 9-97
Connect Initiate Message, 9-89
Connect Reject Message, 9-99
Disconnect Message, 9-106
Interrupt Message, 9-101
Link Abort Message, 9-107
Link Service Message, 9-102
Local Data Message, 9-87
Local Data Message with Privilege Mask, 9-88
Network Data Message, 9-100
Received formats (FORTRAN)
Connect Confirm Message, 10-93
Connect Initiate Message, 10-85
Connect Reject Message, 10-95
Disconnect Message, 10-103
Interrupt Message, 10-98
Link Abort Message, 10-104
Link Service Message, 10-99
Local Data Message, 10-83
Local Data Message with Privilege Mask, 10-84
Network Data Message, 10-96
Received formats (MACRO)
Connect Confirm Message, 11-112

Received formats (MACRO) (Cont.)
Connect Initiate Message, 11-103
Connect Reject Message, 11-115
Disconnect Message, 11-125
Interrupt Message, 11-119
Link Abort Message, 11-127
Link Service Message, 11-121
Local Data Message, 11-99, 11-101
Network Data Message, 11-117
Receiver
ID blocks, 5-12, 6-5
Receiver identity, 2-8, 3-5, 3-9, 8-3
consistency with access, 5-10, 6-8, 9-10, 10-8, 11-12
declaring, 4-2
name, 3-5, 4-2, 5-7, 6-5, 6-25, 6-29, 8-4, 10-5, 11-9, 11-32
network object type code, 3-5, A-1
object type code, 4-2, 5-7, 6-5, 8-4, 9-8, 10-6, 11-10
uniqueness, 4-2
Receiver sleep
awaken from, 5-95
job in, 5-95
RECORDSIZE, 5-2
Remote link status flags, 5-116, 6-101, 9-103, 10-99, 11-122
Remote program
definition, 1-2
identification, 10-36, 10-87, 11-49
Remove Receiver, 3-3, 3-7, 7-4
system call, 5-16, 6-13, 9-15, 10-14, 11-17
RIB, 3-6, 3-7, 4-3, 4-4, 4-11, 5-4, 6-5, 9-7, 10-5, 11-8
RLA, 2-7
Routing, 1-4, 3-3
area nodes, 1-4
hop, 1-1
network diameter, 1-1
path, 1-1
path length, 1-1
Routing node, 1-5
Routing Protocol, 1-4
RSTS/E system monitor, 1-5, 3-1, 8-2, 9-1, 10-1
RSX Run-Time System, 11-6, 11-7
RT11 Run-Time System, 11-6, 11-7

-S-

Segment, 7-3
beginning of message, 5-63, 5-112, 6-53, 6-97, 9-100, 10-53, 11-66
definition, 4-13
end of message, 4-14, 5-63, 5-112, 6-53, 6-97, 9-100, 10-53, 11-66
maximum size, 4-13
request count, 4-14, 5-76, 5-116, 5-117, 6-65, 6-101, 6-102, 9-103, 9-104, 10-64, 11-78
Segment flow control, 4-12
Selective receives, 3-4, 3-14, 3-15, 6-78, 11-92
Send, 3-3
Send calls (BASIC-PLUS)
Connect Confirm Message, 5-50
Connect Initiate Message, 5-36
Connect Reject Message, 5-56
Disconnect Message, 5-80
Interrupt Message, 5-68
Link Abort Message, 5-85
Link Service Message, 5-74
Local Data Message, 5-26
Local Data Message with Privilege Mask, 5-32
Network Data Message, 5-61
Send calls (BASIC-PLUS-2)
Connect Confirm Message: NTCC, 6-44
Connect Initiate Message: NTCI, 6-32
Connect Reject Message: NTCR, 6-49
Disconnect Message: NTDI, 6-69
Interrupt Message: NTIN, 6-59
Link Abort Message: NTLA, 6-73
Link Service Message: NTLS, 6-64
Local Data Message with Privilege Mask: MSPR, 6-28
Local Data Message: MSLD, 6-23

Send calls (BASIC-PLUS-2) (Cont.)
Network Data Message: NTDM, 6-53
Send calls (COBOL)
Connect Confirm Message: NTCC, 9-46
Connect Initiate Message: NTCI, 9-34
Connect Reject Message: NTCR, 9-51
Disconnect Message: NTDI, 9-72
Interrupt Message: NTIN, 9-62
Link Abort Message: NTLA, 9-76
Link Service Message: NTLS, 9-67
Local Data Message with Privilege Mask: MSPR, 9-30
Local Data Message: MSLD, 9-25
Network Data Message: NTDM, 9-55
Send calls (FORTRAN)
Connect Confirm Message: NTCC, 10-44
Connect Initiate Message: NTCI, 10-32
Connect Reject Message: NTCR, 10-49
Disconnect Message: NTDI, 10-68
Interrupt Message: NTIN, 10-59
Link Abort Message: NTLA, 10-72
Link Service Message: NTLS, 10-63
Local Data Message with Privilege Mask: MSPR, 10-28
Local Data Message: MSLD, 10-23
Network Data Message: NTDM, 10-53
Send calls (MACRO)
Connect Confirm Message: NTCC, 11-54
Connect Initiate Message: NTCI, 11-41
Connect Reject Message: NTCR, 11-60
Disconnect Message: NTDI, 11-82
Interrupt Message: NTIN, 11-71
Link Abort Message: NTLA, 11-87
Link Service Message: NTLS, 11-76
Local Data Message with Privilege Mask: MSPR, 11-36
Send calls (MACRO) (Cont.)
Local Data Message: MSLD, 11-30
Network Data Message: NTDM, 11-65
Send/Receive
sender selection summary, 5-97t
Session Control, 1-5
SET EXECUTOR, 4-8
SET OBJECT, 4-6
Sleep
in Declare Receiver, 9-10, 10-8, 11-12
in Receive, 3-4, 9-80, 9-84, 11-92, 11-96
Sleep bit
in Declare Receiver, 5-9, 6-7
Sleep flag
in Receive, 6-78
Sleep monitor call, 5-9, 6-7, 9-10, 10-8
Sleep monitor directive, 11-12
Source program, 2-2
Starting line number, 4-6
Strings
defining, 10-2
in system calls, 6-3
system calls, 5-2
Synchronous I/O, 7-2
SYSIO privilege
declare receiver, 10-7
System buffer space, 2-10, 3-3, 3-7
System calls
arguments, 5-3, 6-2
Declare Receiver, 5-4, 6-5, 9-7, 10-5, 11-8
"$" form, 11-4
"." form, 11-2
general form, 5-1, 6-1, 10-1
general form with concise interface, 8-1
general form with extended interface, 9-1
Get Local Node Parameters, 5-18, 6-15, 9-17, 10-16, 11-20
Log User Event, 5-20, 6-18, 9-20, 10-19, 11-24
macro expansions, 11-1
Receive, 5-90, 5-97, 6-77, 9-79, 10-76, 11-91

System calls (Cont.)
Remove Receiver, 5-16, 6-13, 9-15, 10-14, 11-17
Send Connect Confirm Message, 5-50
NTCC, 6-44, 9-46, 10-44, 11-54
Send Connect Initiate Message, 5-36
NTCI, 6-32, 9-34, 10-32, 11-41
Send Connect Reject Message, 5-56
NTCR, 6-49, 9-51, 10-49, 11-60
Send Disconnect Message, 5-80
NTDI, 6-69, 9-72, 10-68, 11-82
Send Interrupt Message, 5-68
NTIN, 6-59, 9-62, 10-59, 11-71
Send Link Abort Message, 5-85
NTLA, 6-73, 9-76, 10-72, 11-87
Send Link Service Message, 5-74
NTLS, 6-64, 9-67, 10-63, 11-76
Send Local Data Message, 5-26
MSLD, 6-23, 9-25, 10-23, 11-30
Send Local Data Message with Privilege Mask, 5-32
MSPR, 6-28, 9-30, 10-28, 11-36
Send Network Data Message, 5-61
NTDM, 6-53, 9-55, 10-53, 11-65
summary, 3-2
System library, 8-2, 9-1, 9-6, 10-4, 11-7
System manager, 1-6, 2-11, 4-1, 4-3, 4-5, 4-8, 4-9, 9-20, 11-24

-T-

Target program, 2-2
Target string, 5-1
Task Builder (TKB), 10-4, 11-7
Timing
automatic start-up, 4-7
transmit queue management, 4-8
TKB (Task Builder), 10-4, 11-7
TLK/LSN, 1-5
Transmit maximum, 3-10, 3-11, 3-12, 5-39, 5-102, 5-110, 6-33, 6-45, 6-87, 6-95, 9-90, 9-97, 10-33, 11-43
Transmit queue management, 4-8
effect on aborts, 4-9
effect on DECnet/E programs, 4-9
effect on disconnects, 4-9
message limit, 4-9
timing, 4-8
Truncation of received messages, 3-4, 6-78

-U-

ULA, 2-2, 3-15
User data accompanying messages, 3-4, 3-12, 3-13, 3-15
User event
parameter data, 10-20
User events, 3-3, 3-8, 7-4
class and type, 5-22, 6-19, 9-21, 10-20, 11-25
entity, 5-22, 6-19, 9-21, 10-20, 11-26
parameter data, 5-22, 6-19, 9-21, 11-26
USING clause, 9-1

-V-

VALUE IS clause, 8-2, 9-2

-X-

XRB, 11-6, 11-91
input to .MESAG, 12-1

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

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

In Canada call **800-267-6146**

In New Hampshire, Alaska or Hawaii call **603-884-6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: P&SG Business Manager

ELECTRONIC ORDERING

Dial 800-DEC-DEMO with any VT100 or VT200 compatible terminal and a 1200/2400 baud modem.
If you need assistance, call 800-DEC-INFO.

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&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

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809-754-7575

Reader's Comments

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. _____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number. _____

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Office Systems Documentation MK01-2/E02
DIGITAL EQUIPMENT CORPORATION
Continental Boulevard
Merrimack N.H. 03054

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line