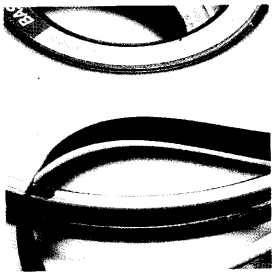
Prime Computer, Inc.

DOC3710-193L
PRIMENET Guide
Revision 19.3

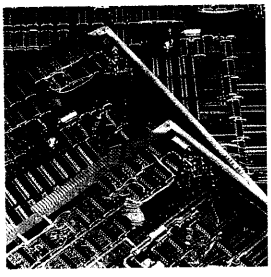













PRIMENET Guide

DOC3710-193

Third Edition

by

Ann Venne and Philip Fulchino

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 19.3 (Rev. 19.3).

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

COPYRIGHT INFORMATION

CREDITS

We are indebted to Bertil Lindblad of Prime, Sweden, and to James Craig Burley and Dave Roberts of Prime, Technical Publications, for their contributions to this book.

PRINTING HISTORY — PRIMENET Guide

Edition	Date	Number	Software Release
First Edition	June 1979	IDR3710	16.3
Update 1	January 1980	PTU2600-065	17.2
Update 2	December 1980	PTU2600-069	18.1
Second Edition	July 1982	DOC3710-190	19.0
Third Edition	February 1984	DOC3710-193	19.3

HOW TO ORDER TECHNICAL DOCUMENTS

U.S. Customers

Software Distribution
Prime Computer, Inc.
1 New York Ave.
Framingham, MA 01701
(617) 879-2960 X2053

Prime Employees

Communications Services
MS 15-13, Prime Park
Natick, MA 01760
(617) 655-8000 X4837

Customers Outside U.S.

Contact your local Prime
subsidiary or distributor.

Contents

ABOUT THIS BOOK xiii

PART I -- AN OVERVIEW OF PRIMENET

1 WHAT IS PRIMENET?

Introduction 1-1
Basic PRIMENET Features 1-2
Remote File Access 1-2
Remote Login 1-3
The File Transfer Service (FTS) 1-3
The NETLINK Utility 1-3
Advanced PRIMENET Features 1-4
PRIMENET Architecture 1-4
Public Data Networks (PDNs) 1-4
Programmer Features 1-5
Ports and Virtual Circuits 1-5
IPCF Subroutines 1-5
FTS Programming 1-6
Operator Features 1-6
Starting and Stopping the Network 1-6
Monitoring Network Servers 1-6
Monitoring Network Events 1-6
Monitoring FTS 1-7
Monitoring RINGNET 1-7

PART II -- ELEMENTARY PRIMENET

2 ACCESSING REMOTE FILES

Introduction 2-1
Remote File Access 2-1
Using Remote IDs 2-2
When to Use Remote IDs 2-3
Establishing Remote IDs 2-4
Examining Your Remote IDs 2-5
Removing Remote IDs 2-5

3 LOGGING INTO REMOTE SYSTEMS

Remote Login 3-1
Error Messages 3-2

PART III — THE FILE TRANSFER SERVICE (FTS)

4 INTRODUCTION TO FTS AND FTR

Introduction to FTS 4-1
Introduction to FTR 4-4
Source and Destination Sites 4-4
File Types 4-5
File Transfer Request Names and Numbers 4-5
FTS and the COPY Command 4-6
Access Rights 4-6
Access Rights for the Requesting User 4-6
Access Rights for FTS 4-7

5 TRANSFERRING FILES WITH FTR

Introduction 5-1
FTR's Help Facility 5-2
Sending a File 5-2
Fetching a File 5-3
Printing a File at a Remote Site 5-3
Checking the Status of Requests 5-4
Using the -STATUS Option 5-5
Using the -DISPLAY Option 5-5
Logging Request Events 5-7
Requesting Notification of Transfers 5-8
Canceling Requests 5-9
If a Transfer Fails 5-10
If the Error Does Not Preclude Trying Again 5-10
If The Error Precludes Trying Again 5-10
Other Options 5-10

6 AN FTR REFERENCE

Introduction 6-1
Options for Submitting Requests 6-1
Summary of Submittal Options 6-2
Full Descriptions of Submittal Options 6-4
Options for Managing Requests 6-13
Summary of Management Options 6-14
Full Descriptions of Management Options 6-14

PART IV -- NETLINK

7 INTRODUCTION TO NETLINK

Introduction 7-1
What is NETLINK? 7-1
NETLINK's Modes of Operation 7-3
  Command Mode 7-3
  Data Transmission Mode 7-4
Network Addresses 7-4
NETLINK File Transfers 7-4

8 USING NETLINK

Introduction 8-1
Invoking NETLINK 8-2
Making a Connection 8-2
Addressing Another System 8-2
  PRIMENET-configured Name Addressing 8-3
  Public Data Network (PDN) Addressing 8-3
  International Addressing 8-3
  Literal Addressing 8-4
  Connect Packet Options 8-4
Direct Remote Login 8-4
Multiple Connections 8-5
Example of the PROMPT Command 8-6
Examples of the FILE and OUTFILE Commands 8-7
  File Transfer Capabilities 8-7
  Precautions 8-7
  Local-to-remote File Transfers 8-8
  Remote-to-local File Transfers 8-9
Running NETLINK from a Command Input File 8-9
Example of the PROFILE Command 8-11
Example of the PAR Command 8-13
Debugging Using NETLINK 8-15

9 A NETLINK REFERENCE

Introduction 9-1
Command Summary 9-1
  Basic Commands 9-2
  Address Formats 9-2
  Profile Commands 9-3
  Other Commands 9-5
Command Reference 9-5
Error Messages 9-17
  Error Messages Generated from PDNs 9-17
  PRIMENET Error Messages 9-19

PART V -- ADVANCED PRIMENET

10 PRIMENET ARCHITECTURE

Introduction 10-1
PRIMENET's Layers 10-1
Advantages of Layered Architecture 10-3
Network Types 10-4
Prime's Local Area Network: RINGNET 10-6
Point-to-point Connections 10-8
Public Data Network (PDN) Connections 10-9
Route-Through Connections 10-9

11 PORTS AND VIRTUAL CIRCUITS

Introduction 11-1
Ports 11-2
Ports, Programs, and User Processes 11-2
Port Assignments 11-2
Virtual Circuits 11-4
Intra-node Calls 11-4
Passing Off Virtual Circuits to Other User Processes 11-5
Virtual Circuit Status Array 11-5
Clearing Codes 11-6

12 PRIMENET AND PDNs

Introduction 12-1
PDNs and NETLINK 12-1
Ease of Access 12-2
Multiple PDN Support 12-2
Route-through 12-2

PART VI -- PRIMENET PROGRAMMING

13 INTRODUCTION TO NETWORK PROGRAMMING

Introduction 13-1
IPCF Subroutines 13-1
IPCF Programming Examples 13-2
IPCF Programming Strategy 13-2
FTS Programming with the FT$SUB Subroutine 13-2

14 IPCF SUBROUTINES

Introduction 14-1
IPCF Overview 14-2
Naming Conventions 14-2
Summary of IPCF Subroutines 14-3
Subroutine Descriptions 14-4
Assigning PRIMENET Port to Receive Incoming Call(s) 14-4
Call Requesting 14-6
Find Information on Incoming Call 14-11
Call Acceptance 14-14
Transmit Data 14-16
Receive Data 14-18
Clear 14-20
Release a PRIMENET Port 14-23
General Network Cleanup 14-23
Wait for Completed PRIMENET Action 14-24
Virtual Call Transfers 14-25
Network Status Interrogations 14-29

15 IPCF PROGRAMMING EXAMPLES

Introduction 15-1
Establishing a Virtual Circuit 15-1
General Layout of Programs Using IPCF Routines 15-2
The Calling Application 15-2
The Called Application 15-2
IPCF Examples 15-3
PRIMENET File-transmission System 15-3
The Transmitting Side 15-4
The Receiving Side 15-6
Routine to Wait for Next Network Event 15-8
Fast Select Calls 15-9
Capacity and Service Busy 15-10
Timing Aspects 15-10
Virtual Circuit Timeout Handling 15-11
The Code 15-11

16 IPCF PROGRAMMING STRATEGY

Introduction 16-1
Front-end Principles 16-2
Server Principles 16-2
Performance Aspects 16-3
Windows and Packet Sizes in Virtual Circuits (Throughput) 16-4
Network Event Waiting 16-5

Checking Return Codes 16-6
Virtual Circuit Clearing 16-7
Program Closedown 16-7
The Effect of START_NET and STOP_NET on IPCF Programs 16-8

17 FTS PROGRAMMING

Introduction 17-1
Program Setup for FT$SUB 17-2
Declaring FT$SUB 17-2
Defining Keys and Error Codes 17-3
Loading the FTS Subroutine Library 17-3
Invoking the FT$SUB Subroutine 17-4
Function Categories 17-5
Transfer Request Submission 17-5
Submission Error Codes 17-8
Transfer Request Modification 17-10
Changing the Status of a Transfer Request 17-14
Status Retrieval of a Transfer Request 17-19
Internal vs. External Names 17-22
Error Codes 17-27
The Request Data Structure 17-28
The Error Data Structure 17-31
Example 17-34

PART VII - OPERATOR'S GUIDE TO PRIMENET

18 MONITORING PRIMENET AND FTS

Introduction 18-1
Monitoring the PRIMENET* Directory 18-2
PRIMENET* Directory Files 18-2
Access Rights 18-3
Adding Remote Disks 18-3
Network Configuration File 18-3
Starting and Stopping PRIMENET 18-4
Monitoring Network Servers 18-4
Monitoring Network Events 18-4
Monitoring FTS 18-5
Monitoring RINGNET 18-5

19 STARTING AND STOPPING PRIMENET

Introduction 19-1
NET ON Versus START_NET 19-2
The START_NET Command 19-2
Invoking START_NET 19-2

The STOP_NET Command 19-3
Invoking STOP_NET 19-4

20 MONITORING NETWORK SERVERS

Introduction 20-1
Monitoring the Network Server Process (NETMAN) 20-1
Monitoring the Route-through Server (RT_SERVER) 20-2
Using Status Users 20-2
Using Status Network 20-3

21 MONITORING NETWORK EVENTS

Introduction 21-1
The PRINT_NETLOG Command 21-2
Network Event Types 21-6
Controlling the Size of the Network Event Log File 21-7
Network Event Messages 21-8
Print NETLOG Error Messages 21-14

22 MONITORING FTS

Introduction 22-1
The FTOP Command 22-2
Summary of FTOP Options 22-2
FTOP Options 22-3
Starting, Stopping, and Monitoring YTSMAN 22-7
Starting, Stopping, and Monitoring File Transfer Servers 22-7
Managing and Monitoring User Requests 22-9
Rush Requests 22-9
Monitoring the FTSQ* Directory 22-10
Monitoring and Archiving FTS Log Files 22-10
Stopping FTS 22-11

23 MONITORING RINGNET

Ring Diagnostic Programs 23-1
RINGNET Overview 23-1
RINGNET Hardware 23-2
RINGNET Terminology 23-2
How RINGNET Works 23-5
The MONITOR_RING Program 23-6
Invoking the MONITOR_RING Program 23-7
Selecting MONITOR_RING Displays 23-8
Interpreting Ring Statistics 23-9
Basic Display 23-10
Error Screen 23-15

Status Line 23-17
Error Display Fields 23-17
Trace Display 23-22
Report File Format 23-23
The FIND_RING_BREAK Program 23-24
How FIND_RING_BREAK Works 23-24
Invoking FIND_RING_BREAK 23-26
The FIND_RING_BREAK Input File 23-26
Creating the FIND_RING_BREAK Input File 23-7
Locating a Break in the Ring 23-28
FIND_RING_BREAK With an Input File 23-29
FIND_RING_BREAK Without an Input File 23-30
FIND_RING_BREAK Error Messages 23-30

APPENDIXES

A X.25 PROGRAMMING GUIDELINES

Introduction A-1
Call User Data Field Terminolgy A-1
X.25 Window and Packet Size A-2
Data Flow Checkpoints A-3

B NETLINK PARAMETERS

International Parameters B-1
TELENET Parameters B-2
Interpreting PAR Command Output B-3

C FTS ERROR MESSAGES

Introduction C-1
General Error Messages C-1
FTGEN Error Messages C-2
FTR Error Messages C-4
FTOP Error Messages C-12

D START_NET AND STOP_NET ERROR MESSAGES

Introduction D-1
START_NET Error Messages D-1
STOP_NET Error Messages D-4

INDEX X-1

About This Book

The PRIMENET Guide provides tutorial and reference information on:

- Remote file access
- Remote Login
- File Transfer Service (FTS) and the File Transfer Request utility (FTR)
- NETLINK (Prime's Packet Assembler/Disassembler (PAD) emulator)
- Ports and virtual circuits
- Inter-Process Communications Facility (IPCF) subroutines

This book is intended for three groups of people:

- Network users who are familiar with PRIMOS commands. Utilities like NETLINK and FTS permit network users to talk to users on other systems and to transfer files required by those users.
- Programmers writing networking programs with Inter-Process Communications Facility (IPCF) subroutines or with the File Transfer Service (FT$SUB) subroutine.
- Operators who monitor and control nodes in a network.

System Administrators who are designing a network, or want their system to become part of a network, should use the Network Planning and Administration Guide (DOC7532-193).

PRIME DOCUMENTATION CONVENTIONS

The following conventions are used in command formats, statement formats, and in examples throughout this document.

Convention	Explanation	Example
UPPERCASE	In command formats, words in uppercase indicate the actual names of commands, statements, and keywords. They can be entered in either uppercase or lowercase.	NETLINK
Abbreviations	If a command or statement format has an abbreviation, it is indicated by an underlining. However, where this cannot be done clearly, the abbreviation either appears below the command with both enclosed in braces, or on a line beneath the format and introduced by the word: "Abbreviation".	QUIT { -DSTN_NTFY } { -DN } -SRC_NTFY Abbreviation: -SN
lowercase	In command formats, words in lowercase indicate variables, items for which the user must substitute a suitable value.	NETLINK -TO nodename
Underlining in Examples	In examples, user input is underlined, but system prompts and output are not.	OK, DATE -MONTH September OK,
Brackets [ ]	Brackets enclose a list of one or more optional items. Choose none, one, or more of these items (0-n).	FTR -HELP [subject] [USAGE]
Braces { }	Braces enclose a vertical list of items. Choose one and only one of these items.	MODE { REMOTE_ECHO } { NO_REMOTE_ECHO }
Ellipsis ...	An ellipsis indicates that the preceding item may be repeated.	item-x[,item-y]...

Parentheses ( )	In command or statement formats, parentheses must be entered exactly as shown.	DIM array (row,col)
Hyphen -	Wherever a hyphen appears in a command line option, it is a required part of that option.	FTR -HELP
(CR)	The (CR) symbol indicates a single carriage return which is generated on most terminals by hitting the RETURN key.	

PART I

An Overview of PRIMENET

1

What is PRIMENET?

INTRODUCTION

This chapter provides an overview of PRIMENET, Prime's networking software. PRIMENET is supported on all Prime 50 Series systems, and provides a reliable, standardized medium through which remote user and file accesses can be made. PRIMENET supports communications between linked systems with a variety of services and transmission methods. PRIMENET makes referencing remote information identical to referencing local information. PRIMENET does not require you to learn new commands to access disks or files on remote systems in a network. Because of this transparency, you do not need to learn new commands or details about system links and locations.

Besides being easy to use, PRIMENET software meets the Consultative Committee for International Telephone and Telegraph (CCITT) 1980 extensions to the X.25 communications standard for packet-switching networks. This allows Prime software to communicate over international public data networks (PDNs) that support X.25. Prime ring and point-to-point synchronous networks are also supported; the combination of network types gives you many options that you can exercise, based on need and application. See Chapter 10 for more information on RINGNET and networks.

To bring up PRIMENET, the System Administrator uses the global network configurator called CONFIG_NET. See the Network Planning and Administration Guide for information on setting up a network.

This chapter briefly describes the following subjects, which are fully described later in this book.

- Basic network user features
- More advanced network user features
- Network programming

BASIC PRIMENET FEATURES

PRIMENET's communications facilities are easy-to-use. The PRIMOS commands you already know can be used across networks. The following sections describe these elementary facilities. Turn directly to later chapters for specific instructions on the use of

- Remote file access
- Remote login
- File Transfer Service (FTS)
- The NETLINK utility

These facilities allow you to login or access files on other systems in a PRIMENET network.

Remote File Access

PRIMENET provides immediate access to any remote file within the network, even if you do not know on which system within the network the file resides. This means that you do not have to learn any new commands to specify a remote file, since PRIMENET works with the file system to access the file in a transparent fashion. In fact, you may not even know that the file is not contained within the local system. In addition, programs accessing remote information do not have to be changed or recompiled if the remote information is moved. Chapter 2 contains additional information on remote file access.

Remote Login

You can log into any remote system connected to the local system through any of the following network types supported by PRIMENET.

- RINGNET
- Point-to-point
- Public Data Network (PDN)
- Route-through connection

Once you are logged into a remote system, you can type commands as if you were logged into the system locally. Chapter 3 has additional information on remote login. Refer to Chapter 10 for a description of the four network types supported by PRIMENET.

The File Transfer Service (FTS)

The File Transfer Service (FTS) allows you to transfer files between Prime systems in a communications network. You can send or fetch any file (given the proper access rights) in the system; if a node is currently disconnected, FTS automatically retries a transfer later.

FTS has separate facilities for users, operators, and System Administrators. For users, the FTR command transfers files. The FTGEN utility is used by the Network Administrator to install FTS on a network. See the Network Planning and Administration Guide for more information on this utility. The FTR utility and the FTOP utility are used by system operators to monitor and expedite user requests submitted with FTR. FTS also includes a subroutine, FT$SUB, that can be used in programs to perform FTR options. Chapters 4 through 6, 17, and 22 contain additional information on FTS.

The NETLINK Utility

NETLINK lets you communicate over any X.25 network to which your local system is linked. NETLINK emulates a PDN Packet Assembler/Disassembler (PAD). It converts the asynchronous terminal output into X.25-formatted packets of information that can be transmitted over an X.25 network.

If your system has a PDN link, NETLINK allows access to any system in that network, both Prime and non-Prime. You do not have to log out of the local system to invoke NETLINK; in fact, NETLINK supports simultaneous links of up to six remote systems and allows you to move between them and the local system at will. This capability puts a variety of PDN facilities in quick reach of any Prime user. Chapters 7 through 9 contain additional information about NETLINK.

NETLINK can be used not only to connect to Public Data Networks and other Prime systems, but for file transfer as well. This is described in Chapter 8.

ADVANCED PRIMENET FEATURES

Advanced PRIMENET features may require an understanding of

- PRIMENET architecture
- Ports and virtual circuits
- Public Data Networks (PDNs)

PRIMENET supports the X.25 standard, and can communicate with other systems and PDNs that support this communications protocol.

PRIMENET Architecture

PRIMENET architecture consists of various software and hardware layers that handle network links between applications, control the flow of data between systems, and provide user facilities. Chapter 10 describes PRIMENET's layered architecture.

Public Data Networks (PDNs)

The 50 Series systems can subscribe to all Public Data Networks (PDNs) that support the CCITT X.25 protocol standard. All of these networks transfer and process information in packets. Packet-switching networks can share the same equipment among several users simultaneously. These networks can often provide service at substantial savings over methods that require dedicated transmission lines or dialup circuits.

The user of a Prime 50 Series system that is linked to a PDN has access to all other members of the PDN. This means that any Prime terminal user can access all other member systems, both Prime and non-Prime, and that all PDN terminal users can access the 50 Series system. Both Prime systems and PDNs administer access controls, of course, for users who desire greater security.

PROGRAMMER FEATURES

The following programming features are provided by PRIMENET:

- Ports and virtual circuits
- Interprocess Communications Facility (IPCF) subroutines
- The FTS FT$SUB subroutine

Applications can make virtual circuit connections to other systems, and accept incoming connections. Programmers can also write file transfer applications with the FT$SUB routine, which invokes FTR options.

Ports and Virtual Circuits

An application that communicates with a remote system must have some way of identifying the remote node and the destination application. The node can be identified through its address in the network configuration. The destination application must be identified through a port. Each node has a list of ports that act as subaddresses within each node. The application making the connection must assign one of these ports to contact the destination application.

When one application specifies the node and port of another application, PRIMENET establishes a connection between the two through a virtual circuit. This is a logical path or channel that traverses the network from one application to another through any of several physical, point-to-point links. For information on ports and virtual circuits, see Chapter 11.

IPCF Subroutines

Interprocess Communication Facility (IPCF) subroutines allow an application to set up communication links with target applications within the network. Through these links, the application can exchange data with any of its target applications.

Procedures written in any high-level language can call any of the IPCF subroutines. This capability is especially useful when you are developing distributed applications. Chapters 13 through 16 contain detailed information on IPCF subroutines.

FTS Programming

Programmers can write applications to call FTS through the FT$SUB subroutine. FT$SUB invokes FTR options that submit, modify, cancel, abort, hold, and release file transfer requests. This subroutine is described in Chapter 17.

OPERATOR FEATURES

Operators monitor and control network operations with the following tools.

- START_NET and STOP_NET
- STATUS USERS and STATUS NETWORK
- PRINT_NETLOG
- FTR and FTOP
- MONITOR_RING and FIND_RING_BREAK

These tools help the operator to monitor and control the various user facilities that run in a network environment.

One of the responsibilities of an operator is to monitor certain system directories (for example, the PRIMENET* directory) that are used and created by network facilities. This information is described in Chapter 18.

Starting and Stopping the Network

Two new commands, START_NET and STOP_NET, allow you to add or remove a system from your network without interrupting local system activity. See Chapter 19 for more information.

Monitoring Network Servers

Network servers can be monitored with the STATUS USERS and STATUS NETWORK commands. Chapter 20 describes how servers are monitored.

Monitoring Network Events

PRIMENET logs network events in a log file when network event logging has been turned on (with the system configuration directive NETREC or with the EVENT_LOG -NET ON command).

The PRINT_NETLOG command collects these events in a file called NETLST, or lists the file on the terminal. This command is described in Chapter 21 on monitoring network events.

Monitoring FTS

The File Transfer Request utility (FTR) has a set of management options that let an operator manage and control user-submitted file transfer requests.

The FTS Operator utility (FTOP) lets operators who are logged in under the user-id SYSTEM start and stop the file transfer manager and the file transfer servers. Chapter 22 describes FTR and FTOP as they pertain to operator responsibilities.

Monitoring RINGNET

You can monitor RINGNET with two programs:

- MONITOR_RING
- FIND_RING_BREAK

MONITOR_RING displays throughput and error statistics of ring traffic on the node that it is run on.

FIND_RING_BREAK shows you the location of breaks in the ring. This program works only for hardware failures that cause complete signal interruption on the ring. FIND_RING_BREAK can be useful in conjunction with MONITOR_RING to determine a failure. These programs are described in Chapter 23.

PART II

Elementary PRIMENET

2
Accessing Remote Files

INTRODUCTION

This chapter explains how to use PRIMENET to access files that are physically located on a disk on a remote computer.

Explanations of how the System Administrator sets up remote file access are in the Network Planning and Administration Guide. Daily network maintenance is described in Part VII of this book. Daily system maintenance is described in the System Operator's Guide, Volumes I and II.

REMOTE FILE ACCESS

Accessing files on a Prime computer that is networked to another computer is normally the same as accessing files locally. The STATUS DISKS command shows which disks are available, and commands such as ATTACH, COPY, DELETE, ED, and LD work on remote files in exactly the same way they do on local files.

Here is an example of editing a file LOANS>UNCOLLECTED, on the disk SHARK, on the system SYSD. Except that STATUS DISKS shows the disk SHARK to be on system SYSD, there is no indication that remote file access was involved. No special remote file access commands were required.

```
OK, STATUS DISKS

Disk    Ldev   Pdev   System
STATS      0   3462
FIELDS     1    460
MISCEL     2  71063
FOREST     3  71061
REEFS      4    460   SYSC
LAGOON     5    460   SYSD
SHARK      6  12060   SYSD
SHARK2     7  52061   SYSD
CLOUDS    12    460   SYSE
CLIFF1    13  12460   SYSE
CLIFF2    14  61461   SYSE
AERIE     15    462   SYSE
ROCK      23  21460   SYSA
FALCON    24  71061   SYSA
NEST1     25    660   SYSA
NEST2     26  10660   SYSA

OK, ED <SHARK>LOANS>UNCOLLECTED
EDIT
L HARRY
Harry W. Sloth    $38.14
C/38/18/
Harry W. Sloth    $18.14
FILE
<SHARK>LOANS>UNCOLLECTED
OK,
```

In the following instances, however, special remote file access commands or actions may be required.

- For security reasons, the remote system does not recognize your user-id as valid for file access. In this case, you need to use a remote-id, as described in the section on USING REMOTE IDs, below.

- The remote disk you wish to access may not be available on your system. In this case, you must either ask an operator or your System Administrator to add the disk, or use the File Transfer Service (FTS) to transfer the file in question to your system.

USING REMOTE IDS

Prime systems linked through PRIMENET usually recognize the user-ids found on any remote system in the network. This means that you can work on a remote system by giving the same commands you would use on the local system. What actually happens is that a special kind of phantom called a slave uses your user-id and does the work on the remote system for you. This process is invisible to you.

The Network Administrator of a Prime computer may configure the system to "force user validation" with respect to other Prime systems. A system that forces user validation does not automatically recognize your usual user-id. The slave cannot do your work on that system. You must establish a remote-id for the remote system. The slave then uses this remote-id. You must create a remote-id to access any remote system that forces user validation.

Using a remote-id involves performing the following steps:

1. Have the System Administrator on the remote system create a user-id for you on that system. This needs to be done only once.

2. Add the remote-id by using the ADD_REMOTE_ID (ARID) command. You need to do this every time you log on and intend to access the remote disk.

Note

You might also wish to establish remote-ids even on systems on which your user-id is recognized. For example, a remote-id may allow you certain ACL rights that your own user-id does not allow.

When to Use Remote IDs

The following is an example of a circumstance requiring the ADD_REMOTE_ID (ARID) command. Two Prime systems, SYSA and SYSB, are connected together by PRIMENET. Each system already has a number of established user-ids. Unfortunately, several user-ids on SYSB are identical to user-ids on SYSA, but belong to different people. For example:

System	User-id	Person
SYSA	RUTH	Ruth J. Morton, Payroll Dept.
SYSB	RUTH	Frank W. Ruth, Sports Director

Company management intends that Ruth Morton be able to access files on SYSA and SYSB; among others, the directory <BURSAR>PAYROLL on SYSA. They also intend that Frank Ruth, who logs in on SYSB, be able to access <BURSAR>BUDGETS>SPORTS on SYSA, but not the payroll.

If management did not set up any special protection, Frank Ruth, logged in as RUTH on SYSB, would be able to access any files on SYSA to which RUTH (Ruth Morton) had access. To avoid this problem, the Network Administrator requires that SYSA "force user validation." This means that RUTH from SYSB is not allowed access as RUTH on SYSA. In order to

access files on SYSA, the SYSB user must supply an additional user-id and password, by which he is recognized on SYSA. For example, suppose Frank Ruth is given the user-id FRANK on SYSA, and the ACL for <BURSAR>BUDGETS>SPORTS provides appropriate access rights to FRANK. In that case, RUTH (Ruth Morton) does not have access to <BURSAR>BUDGETS>SPORTS. And FRANK does not have rights to access <BURSAR>PAYROLL.

To log in on SYSB and access the sports budget (on SYSA), Frank Ruth would use commands such as these:

```
OK, LOGIN RUTH

RUTH (user 50) logged in Tuesday, 7 Feb 84 9:32:02.
Welcome to PRIMOS version 19.3
Last login Monday, 6 Feb 84 13:21:16.

OK, ARID FRANK SECRET -ON SYSA
OK, A <BURSAR>BUDGETS>SPORTS
OK, ED FIRST.QUARTER
. . .
etc.
```

Establishing Remote Ids

You can establish a remote-id with the ADD_REMOTE_ID command, which has the following format.

```
ADD_REMOTE_ID remote-id [password] -ON system [-PROJECT project-id]

Abbreviation: ARID
```

The arguments and options for this command are the same as those for the LOGIN command.

The remote-id is the user-id that the slave uses for you on the remote system, specified by system. In order to be useful, remote-id must already have been defined as a valid user-id by the System Administrator on the remote system.

You must also supply any password or project-id required for access to the remote system. If the remote-id does not exist on the remote system, or if any required password or project id is missing or incorrect, any attempts to access the remote system fail.

You can have up to 16 remote-ids, but only one for any given remote system. For example, if you add the remote-id JINKS on SYSA and then add the remote-id LYNX on SYSA, LYNX replaces JINKS. All remote-ids are removed when you log out.

Examining Your Remote IDs

With the LIST_REMOTE_ID command, you can examine the existing remote ids you have established. The format is:

```
LIST_REMOTE_ID [-ON system]

Abbreviation: LRID
```

If you use the -ON option, only the remote-id for system is listed. If you omit the -ON option, all of your remote-ids are displayed. Passwords are never displayed. For example:

```
OK, LIST_REMOTE_ID

System  User id                    Project id
------  -------                    ----------
SYSB    JIM
SYSK    JAKE                       SALES
SYSM    JODY
OK,
```

Removing Remote IDs

You can also remove remote-ids from your list, if the list gets too large. The PRIMOS command is REMOVE_REMOTE_ID.

```
REMOVE_REMOTE_ID [-ON nodename]

Abbreviation: RRID
```

The REMOVE_REMOTE_ID command allows you to remove your remote-id for a given system from your remote-id list. You may have to remove a seldom-used or obsolete remote-id to add a new remote-id to your list.

If you do not have a remote-id for the system named in "nodename", the message "Not found" appears.

3

Logging Into Remote Systems

REMOTE LOGIN

Logging in through PRIMENET is called remote login because you log into a computer that is remote from the local system to which your terminal is directly connected. PRIMENET provides two methods of remote login.

The first method of remote login uses the familiar LOGIN command, which is described in this chapter. You can use it to log on to a Prime system that is connected in a network and to which you have access rights.

The second method of remote login uses NETLINK, a Packet Assembler/Disassembler (PAD) emulator that lets you connect to networked Prime systems or to non-Prime computers over a Public Data Network (PDN). NETLINK is described in Chapters 7, 8, and 9.

To log in to the remote computer named system, use the following LOGIN command format:

```
LOGIN user-id [password] [-PROJ project_id] -ON system
```

Except for the -ON option, logging in remotely is identical to logging in locally. When you use NETLINK to log into a remote system, however, you must first be logged into the local system.

Here is an example of a successful remote login:

```
LOGIN HANSEN -ON SYSB
PRIMENET 19.3 SYSB

HANSEN (user 50) logged in Saturday, 29 Oct 83 10:12:12.
Welcome to PRIMOS version 19.3
Last login Monday, 24 Oct 83 7:41:16.

OK,
```

You must have a valid user-id on a remote system in order to log in. If your terminal is already logged in, you might get an error message requesting that you log out first. To learn which remote systems are accessible from your terminal, ask your System Administrator or log into your local system and type STATUS NETWORK. (You are not necessarily allowed to log into all the systems displayed by this command, however, since remote login is configured by your System Adminstrator.)

ERROR MESSAGES

The following error messages might occur when you attempt to log into a remote system.

- Please log out first. (RLOGIN)

Log out of your current system before trying to log into another system.

- Invalid system name. (RLOGIN)

You have attempted to log into a disallowed or nonexistent system, or you have spelled the system name incorrectly.

- Remote login to that system not enabled. (RLOGIN)

Users do not have permission to attempt remote login to that system.

- No more remote lines available. (RLOGIN)

All of the available slots for users logging through to a remote system are in use. Only 64 users can log through to remote systems simultaneously.

- Can't connect to remote system. (RLOGIN)

For some unknown reason, calls to the remote system are unsuccessful.

- Line to remote system down. (RLOGIN)

All available links to the remote system are down.

- Too many network calls for link. (RLOGIN)

A network link is currently handling the maximum number of calls allowed. You must wait until someone finishes using the network.

- Too many network calls. (RLOGIN)

Your system is currently handling the maximum number of calls it can support. You must wait until someone finishes a call.

- Invalid facilities. (RLOGIN)

Your system has attempted to send an illegal connect packet for the network to which it is connected. This error message indicates that an improper set of facilities is configured. Inform your System Administrator.

- Can't send login line. dddddd< RCODE dddddd< ERR

Remote login could not send the login line to the remote host because of an internal error. dddddd are decimal numbers for the internal state variables RCODE and ERR. Please refer this information to your support staff.

- Remote system busy.

The remote system has all available circuits in use. Wait for one of the circuits to be freed.

- Remote system down.

The remote system is not responding to messages from the network. It is presumed not to be operating.

- Remote system not up yet.

The system has just been started but is not yet ready to accept calls. This may be because the date and time have not been set yet. Wait a reasonable period for the operator to set the date and time, and try again.

- Remote system has too many calls in progress.

The remote system is not accepting any more remote logins since all available resources are in use. You must wait until resources are freed up.

- No more remote lines on remote system.

The maximum number of allowed remote login users are already logged into the remote system. You must wait for one of these users to log out. If this occurs frequently, you should see if the System Administrator of the remote system can configure more remote login users.

- CALL CLEARED
  CLEARING CODE =:cccccc
  DIAGNOSTIC CODE =:dddddd

Remote login has received an unexpected clearing cause or diagnostic from the network. cccccc is the clearing cause in octal, and dddddd is the diagnostic code in octal. If this problem recurs, you should contact your support staff.

PART III

The File Transfer Service (FTS)

4

Introduction to FTS and FTR

INTRODUCTION TO FTS

This chapter describes the File Transfer Service (FTS), which lets you transfer files between Prime computers. FTS functions over any PRIMENET link, including RINGNET, synchronous links, and also over Public Data Network (PDN) links, such as TELENET or DATAPAC. FTS is made up of the following utilities:

- File Transfer GENeration (FTGEN)
- File Transfer Request (FTR)
- File Transfer OPerator (FTOP)

FTGEN is described in the Network Planning and Administration Guide. FTR is introduced in this chapter. An FTR tutorial is provided in Chapter 5, and a full reference for FTR options is provided in Chapter 6. FTOP is described in Chapter 22. Appendix C describes each FTS error message that might occur in any of the above utilities.

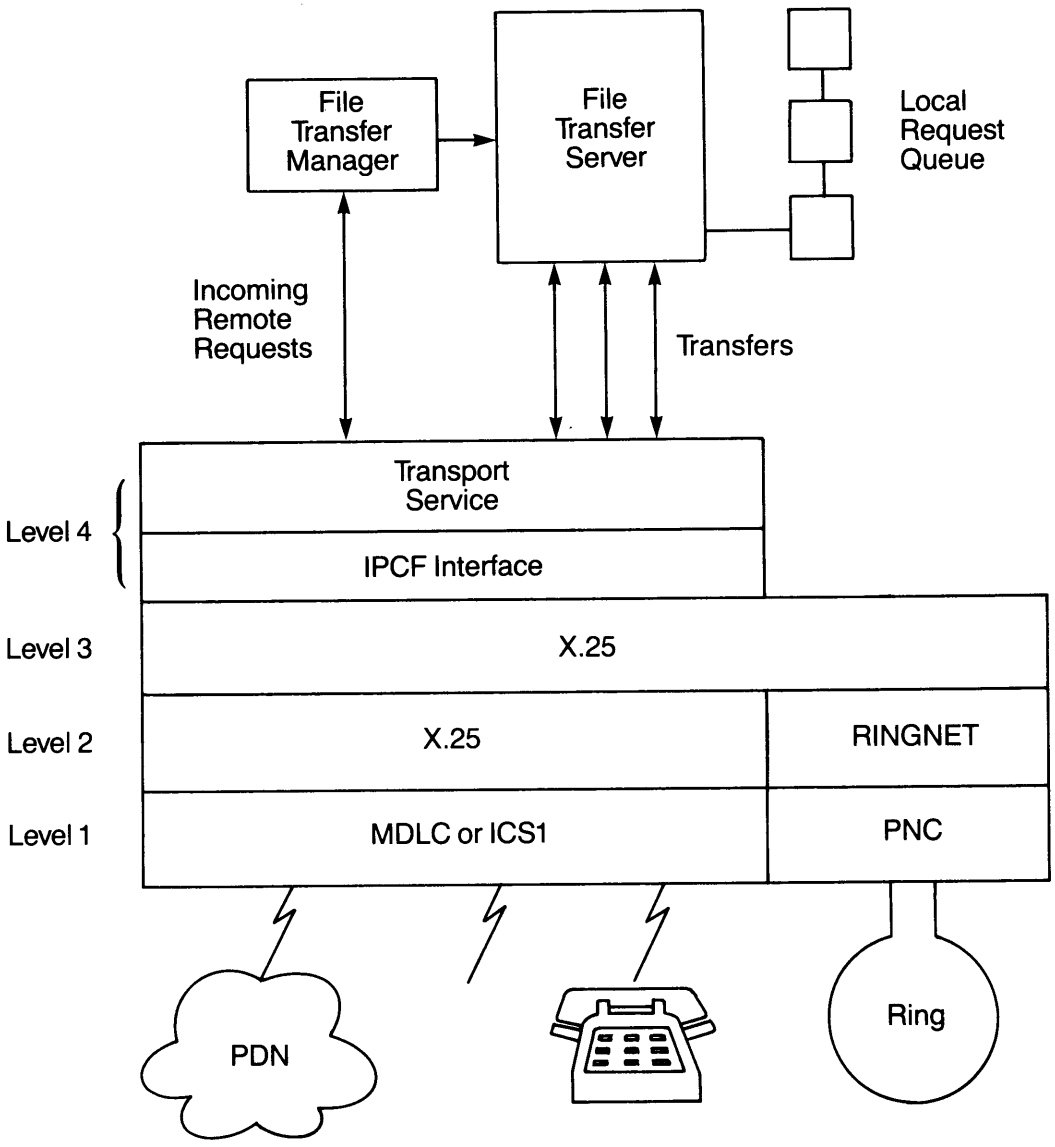
FTS can also be called from application programs through the FT$SUB subroutine. A program can submit, modify, cancel, or otherwise control a request just as a user can with FTR. Chapter 17 describes this subroutine.

In order to use FTS on your system, your System Administrator must install and configure it. PRIMENET must be configured on the system in order for FTS to run. Your System Administrator uses FTGEN to configure one or more file transfer servers, which are phantom processes started from the supervisor terminal at system startup. Each file transfer server has a unique name that helps the operator identify it.

FTS requires a minimum of two phantom processes. One phantom is for YTSMAN, the File Transfer Manager, and one is for a file transfer server. Each server process is responsible for one file transfer queue. A file transfer queue holds file transfer requests made by you and other FTS users. A server process can be stopped and restarted by an operator using FTOP, if necessary. See Chapter 22 for more information on FTOP.

A file transfer server can have password associated with it as a security measure. The System Administrator should set passwords at FTS configuration with FTGEN. Remote FTS sites need to include server passwords in their FTS configurations.

Using FTR, you can request that files be transferred from a local or remote site, or between different directories on a local site. The local site is the node you issue the FTR command on, and the remote site is the other site. Transfers can occur from a local to a remote site or from a remote site to a local site. Local and remote sites are described more fully later in this chapter.



The File Transfer Service (FTS)
Figure 4-1

INTRODUCTION TO FTR

The File Transfer Request (FTR) utility provides a method of transferring files between networked Prime computers and between Prime computers connected to each other through Public Data Networks (PDNs). FTR is one of the FTS utilities.

To transfer a file, use the FTR command to submit a file transfer request. A file transfer request provides FTS with the information that is needed to make a file transfer. That information includes a source-pathname, a destination-pathname, a destination-site, and a destination-user.

You can submit a file transfer request (interactively or by using a CPL file) even when the communications link between two Prime computers is not operational or when the remote computer is down. File transfer requests are queued on the local (requesting) computer.

Once you have submitted a file transfer request, you can display, modify, or cancel it. See Chapter 5 for examples on how to do this.

FTR uses the name of the file being transferred as the name of the request. FTR also gives each request a unique number. These numbers are sequentially assigned to transfer requests to allow unique identification of each request. Request names and numbers are described later in this chapter.

Before you can use FTR, you need to know the following information, which is described in this chapter.

- How to define source and destination sites
- What file types you can transfer
- How to use file transfer request names and numbers

Source and Destination Sites

File transfers take place between sites. A site is a single Prime computer, identified by a unique site name and a server name. Prime sites normally use their PRIMENET system names as site names. Files are transferred from a source site to a destination site. One of these must be your local site; the other is usually a remote site. (FTR cannot be used to transfer files between two remote sites.)

Chapter 5 assumes that you are using FTR between Prime machines that have been configured by the System Administrator with the FTGEN command. To transfer files to or from sites that are not configured, see the descriptions of the -DSTN_SITE and -SRC_SITE options under OPTIONS FOR SUBMITTING REQUESTS in Chapter 6.

File Types

FTS Rev 2 supports transfers of sequentially-accessed (SAM) and directly-accessed (DAM) files and now also allows you to transfer standard SEGSAM and SEGDAM segment directories.

FTS Rev 1 can only transfer SAM and DAM files. FTS Rev 1 also makes transferred SAM files into DAM destination files if the files do not exist at the remote site.

The following table shows FTS dependencies and PRIMOS Revs:

FTS Rev.	PRIMOS Rev. Released On	Will Run On
1.0	18.4	18.2 and above
1.0	19.0	18.2 and above
1.1	19.1, 19.2	19.0 and above
2.0	19.3	19.0 and above

Note

FTS does support transfers of MIDASPLUS and SEG run files, but it does not support transfers of segment directory-based file structures like ROAM, DBMS, or RBF files.

File Transfer Request Names and Numbers

Each file transfer request has associated with it a name and number. The name is usually the file name being transferred, but it can be a name you specify with the -NAME option. The length of a request name is limited to 32 characters.

The number is a sequential number that is assigned by FTS to the request in the order of transfer submission. Leading zeros are suppressed.

You can refer to either the name or the number in FTR command lines. The request number is handy if you have two requests with the same name. Thus, you can use a name when you have only a single request pending, or when you want information on all requests with that name. You can use a number when you want to be sure you're specifying only one request.

FTR AND THE COPY COMMAND

The FTR command or the COPY command can be used to transfer files between networked Prime computers. (The NETLINK utility can be used to transfer files as well. See Chapter 8 for more information.) The two commands give the following alternative approaches (one command or the other can be more appropriate in a given situation).

- COPY provides immediate, direct access to a remote file system. FTR provides a queued file transfer.
- COPY ties up your terminal when the file is being copied. FTR leaves your terminal free during a file transfer.
- COPY requires the remote site to be up and the communications link to be working at the time you request a remote file copy. Otherwise, you receive an error and you must try again later. FTR accepts requests for file transfers at any time, regardless of the state of the remote site or the communications link. FTR queues the requests and transfers files at a time when the remote site and the communications are functioning.
- You can use COPY and other PRIMOS commands for remote file access only when the System Administrator has enabled remote file access communications between the systems. If the communications link is expensive or scarce, FTR may be a good alternative. FTR needs to be connected only long enough to handle queued file transfer requests.

ACCESS RIGHTS

In order for FTS to work correctly, both you and FTS need certain file access rights, since FTS is subject to ACL security mechanisms in the same way as other PRIMOS users and phantoms.

Access rights are any combination of the following rights: Protect (P), Delete (D), Add (A), List (L), Use (U), Read (R), and Write (W). The Prime User's Guide has information on setting ACLs in general.

Access Rights for the Requesting User

To send files, you need LURW access to the source directory. To fetch files, you need ALURW access to the destination directory.

Access Rights for FTS

File transfer servers need access rights as well. They need DALURW to the directory containing the source file and to the directory receiving the file.

You may wish to restrict access rights that you give to FTS servers by creating a subdirectory with full access rights for use by a remote site in transferring files.

Ask your System Administrator for the user-id of the file transfer server(s), so that you can give it appropriate access rights.

You should realize that:

- Normally, FTR takes a copy of the file using your ACL rights. If you use -NO_COPY (-NO_COPY is described in Chapter 6), the FTS server will read the file later. It must have the correct ACL rights. If you specify -DELETE (-DELETE is described in Chapter 6), the server should also have delete rights to the file. If the transfer should fail for any reason, including the lack of appropriate rights, the file will not be deleted.

- When a file transfer server writes or reads a file from a remote site, the remote server must have the appropriate access rights to that file. If the file is being created, the server must have create rights for the appropriate directory. Again, the transfer fails if the FTS does not have the appropriate rights needed to copy a file.

5

Transferring Files With FTR

INTRODUCTION

This chapter tells you how to use the File Transfer Request utility (FTR) to transfer files between Prime computers. You learn how to

- Submit a file transfer request
- Send a file
- Fetch a file
- Print a file
- Check the status of a request
- Cancel a request

You should read Chapter 4, Introduction to FTS and FTR, before using the FTR command. Chapter 6, An FTR Reference, fully describes each FTR option.

FTR'S HELP FACILITY

To obtain a list of FTR HELP subjects, when you are at PRIMOS command level, type either

```
FTR -HELP SUBJECTS
```

or simply,

```
FTR -HELP
```

SENDING A FILE

To send a file to another Prime computer, use the FTR command with the following format:

```
FTR source-pathname destination-pathname -DSTN_SITE sitename

Abbreviation for -DSTN_SITE:  -DS
```

The source-pathname is the pathname of the file being sent. You may give a filename if it is in your current directory.

The destination-pathname specifies the name and location of the transferred file at the destination site.

Note

If the pathname for a source file or destination file contains a passworded directory, the password must be included. The pathname must also be surrounded in single-quote marks, for example, 'MARPLE CLUE>EVIDENCE'. You can type the password in either upper- or lowercase.

The -DSTN_SITE sitename option specifies the name of the destination site. FTR provides a response to your request in the following format:

```
Request request-name (request-number) submitted.
```

The request-name usually is the name of the file you transfer. You can specify a different name with the -NAME option. The request-number is the unique identification number assigned by FTR. For example, assume your local site is named SYS2. The following command line illustrates a file transfer request:

```
OK, FTR CENTER>REPORT EXPOST>GROUP2>TEXT -DSTN_SITE SYS4
[FTR rev 2.0]
Request REPORT (21) submitted.
OK,
```

In this example, FTR queues a copy of the file REPORT from the UFD CENTER to send to system SYS4 for deposit in the directory EXPOST>GROUP2 under the name TEXT. The request name is the source filename, REPORT. The unique request number is 21.

FETCHING A FILE

To fetch a file from another Prime computer, use the following command format:

```
FTR source-pathname destination-pathname -SRC_SITE sitename

Abbreviation for -SRC_SITE:  -SS
```

The source-pathname and destination-pathname are used as defined above for sending a file.

The -SRC_SITE sitename option specifies the name of the site where the file you want is stored.

For example, assume you are on SYS2, and you type the following command:

```
FTR PEOPLE>LIST MYUFD>MYLIST -SRC_SITE SYS6
```

This copies the file LIST in the UFD PEOPLE on SYS6 into the UFD MYUFD on SYS2 under the name MYLIST. The request name is the source filename, LIST.

Note

Keep in mind that the destination file will actually arrive some time after you type in the FTR command. If you attempt to use the file immediately after using FTR, the file will not be found or will contain old data. Use the FTR -STATUS command, described below, to determine when the transfer has completed.

PRINTING A FILE AT A REMOTE SITE

The -DEVICE LP option prints the specified file on the default line printer at the remote site, which is specified by -DSTN_SITE sitename. This option uses the PRIMOS spooler. To print a file at a remote site, use the following command format:

```
FTR source-pathname -DSTN_SITE sitename -DEVICE LP -DSTN_USER name
```

The -DSTN_USER name option specifies the name of the person who should receive the printout at the remote site. The file is printed with the name of the file transfer server (set by your System Administrator) on the first line of the banner instead of a user id. The name specified after -DSTN_USER appears on the second line of the banner of the printed file. The user's name doesn't have to be a user id.

For example, assume you are on a system named SYSA. The following command causes the file LETTER to be printed at the default line printer on the remote system SYSF. You cannot set any SPOOL options. FTS uses the following defaults: the first line of the banner is the user-id of the FTS server, for example, "FTP". The second line has the name of the destination user, which is, in this case, JUDY_JONES.

```
FTR STUART>LETTER -DSTN_SITE SYSF -DEVICE LP -DSTN_USER JUDY_JONES
```

CHECKING THE STATUS OF REQUESTS

Once you have submitted a request, you can check its status with either the -STATUS or -DISPLAY options. Their formats are as follows:

```
FTR -STATUS [ request-name   ]
            [ request-number ]

FTR -DISPLAY [ request-name   ]
             [ request-number ]
```

These options return information on each of your file transfer requests identified by request-name or request-number. If you omit the request name, you receive a report on all of your current requests.

The -STATUS option produces a one-line summary for each request. The -DISPLAY option reports complete information on each request. Both options report the status of a request. This tells you whether the request is being processed, is still awaiting processing, or on hold. The status category is one of the following:

Status	What It Means
Waiting	The request is in the transfer queue.
Transferring	The transfer is in progress.
Aborting	An active transfer request is aborting.
Put on hold by user	The request is being held in the transfer queue.

Put on hold by operator	The request is being held in the transfer queue.
Put on hold by FTS	The request is being held in the transfer queue.

Using the -STATUS Option

This option produces a one-line summary on each request containing the following information.

- The date and time the transfer request was first submitted.
- The user-id of the submitting user — normally, your user-id.
- The request name.
- The request number.
- The transfer queue name.
- The status of the request.

The following shows an example of the STATUS option.

```
OK, FTR -STATUS
[FTR rev 2.0]
83-03-30.10:52:12 ELLEN CHAPTER (36) (FTS$1) Status -    waiting
OK,
```

In this example, ELLEN has a request named CHAPTER waiting to be transferred.

Using the -DISPLAY Option

This option produces a full report on the status of your requests. The display takes the following form.

Category	Information on the Request
Request	Request-name (request-number).
User	Submitter's user-id.
Queue	Queue name where the request is queued.
Queued	Date and time the request was submitted and the request status.

Last attempt	Date and time of the most recent transfer attempt and the number of transfer attempts.
Current time	Current date and time.
Source file	Source pathname.
Source file size	Number of bytes. Displayed only if the source file is on the local site.
Destination file	Destination pathname.
Source site	Source sitename.
Destination site	Destination sitename.
Request log file	Pathname of log file; not always displayed.
Log message level	Level of detail entered in the request log file.
Source user	A user-id (or another name) at the source site to be associated with the transferred file. Not always displayed; useful when notifying a user at a remote site about a transfer.
Destination user	A user-id (or another name) at the destination site to be associated with the transferred file. Not always displayed; useful when printing files at remote sites or when notifying a user at a remote site about a transfer.
Source file type	The type of file being sent or fetched from the source site. The file type is not always displayed.
Destination file type	The type of file being specified for the destination file on the destination site. The file type is not alays displayed.
Options	List of request options.

For example, say your user-id is SMITH on the system OAK, and you want to transfer a file to JONES on the system LINDEN. The following example shows a request being submitted and then displayed.

```
OK, FTR LECTURE <LEAF>JONES>LIST -DSTN_SITE LINDEN -DSTN_USER JONES
[FTR rev 2.0]
Request LECTURE (1) Submitted.
OK, FTR -DISPLAY
[FTR rev 2.0]
Request          - LECTURE (1)
User             - SMITH
Queue            - FTS$1
Queued           - 83-11-01.09:55:39  Status   - transferring
Last attempt     - 83-11-01.09:56:00  Attempts -      1
Current time     - 83-11-01.09:56:07
Source file      - <BRANCH>SMITH>LECTURE
Source file size -        5162 bytes.
Destination file - <LEAF>JONES>LIST
Source site      - OAK
Destination site - LINDEN
Source user      - SMITH
Destination user - JONES
Source file type             - SAM
Destination file type        - SAM
Options :-
BINARY, COPY, NO DELETE, NO SOURCE NOTIFY, NO DESTINATION NOTIFY.
OK,
```

LOGGING REQUEST EVENTS

You can create an automatic log of your file transfer request events by specifying the -LOG pathname, when you submit a request. FTR records logging information in a file named pathname on the system originating the request. If the file named pathname already exists, the logging information is appended to the end of that file. For example, you might give the following command.

```
FTR REMIND <LEAF>JONES>NOTE -DSTN_SITE OAK -DSTN_USER JONES -LOG
LOGFILE
```

If the file transfer of was successful, the entries in the log file, with a normal message level, would look like this:

```
11.29.14: [1.1] Request REMIND (1) started Tuesday, November 1, 1983
11.29.14: [1.1] Submitting user is SMITH
11.29.14: [1.1] Local file is <BRANCH>SMITH>REMIND
11.29.20: [1.1] RESULT: Transfer terminated: Satisfactory and Complete.
11.29.21: [1.1] Request REMIND (1) finished.
```

Note

If the pathname for a log file contains directory passwords, the passwords must be included in the pathname, and the whole pathname must be included in single-quote marks. For example, 'SMITH STARRYEYED>LOGFILE'.

You can increase the detail entered in your log file if you wish by specifying -MESSAGE_LEVEL on the FTR command line with one of the following arguments: DETAILED, STATISTICS, or TRACE. NORMAL (1) is the default message level. The number in brackets represents the file server number (1) and the message level (1). You must also specify the -LOG option. More information on the -MESSAGE_LEVEL option appears in Chapter 6.

REQUESTING NOTIFICATION OF TRANSFERS

You can receive notification of the progress of a submitted request in two ways: through the PRIMOS MESSAGE facility, or through the FTR -LOG option. The -LOG option is a surer way to get notification, since the results are sent to the file you specify. You may not always see all of the MESSAGE notifications if you are logged off, or away from your terminal. However, if you want to avoid creating a log file, you can depend on the MESSAGE facility. To do so, specify, on the FTR command line, one of the following options:

Option	Description
-SRC_NTFY -SN	Notifies the source users when the file transfer starts and ends. You use source notify when you transfer a file.
-DSTN_NTFY -DN	Notifies the destination user when the file transfer starts and ends. You use destination notify when you transfer a file.

You will receive the following messages at your terminal:

```
OK, FTR NEWS IDEAS>PROJECT -DSTN_SITE SYSC -SRC_NTFY
    [FTR rev 2.0]
    Request NEWS (35) submitted.

    OK,
     .
     .
     .
    ***FTSERV (user 109 on SYSB) at 16:41
    Request NEWS (35) transfer started.

    OK,
     .
     .
     .
    ***FTSERV (user 109 on SYSB) at 16:43
    Request NEWS (35) transfer ok.

    OK,
```

In this example, the first message from FTSERV indicates that the file transfer has begun. The second message indicates that the transfer was completed successfully. FTSERV is the name for the FTS server. This name is set by the System Administrator and may be different on your system.

Note

You will only receive these messages when PRIMOS displays the OK, prompt. These are deferred messages.

You can specify both the -SRC_NTFY and -DSTN_NTFY options together on the same command line if you wish to inform a remote source user or remote recipient, as well as yourself, of the progress of the transfer. In this case, you must also identify the remote user to FTR by specifying at least one of the following options on the command line.

Option	Description
-DSTN_USER user-id -DU	Required with -DSTN_NTFY when you are sending a file so that FTR knows whom to notify.
-SOURCE_USER user-id -SU	Required with -SRC_NTFY when you are getting a file so that FTR knows whom to notify.

For example, if the previous example had included the source notification option -SRC_NTFY, the messages listed above are sent to both you and JOHN on SYSC.

```
OK, FTR NEWS IDEAS>PROJECT -DSTN_SITE SYSC -DSTN_USER JOHN
-DSTN_NTFY -SRC_NTFY
```

CANCELING REQUESTS

If you have submitted a request that is currently waiting in a file transfer queue, you can cancel the request with the following command:

```
FTR -CANCEL [ request-name   ]
            [ request-number ]
```

For example, if the request you wished to cancel was named NEWS (12), either of the following commands would cancel the request.

```
FTR -CANCEL NEWS

FTR -CANCEL 12
```

You will receive the following message after you type the above command:

```
Request NEWS (12) cancelled.
```

You cannot cancel requests that are in the process of being transferred.

IF A TRANSFER FAILS

If FTS is unable to complete your transfer request, it takes one of two actions, depending upon the reason for transfer failure.

If the Error Does Not Preclude Trying Again

FTS tries to transfer again. For example, the remote computer may be down, but will likely be up later. The FTS server attempts retransmission every 30 minutes for 72 hours, for a total of 144 tries. If transmission still fails, the request is put on hold with a retry count of 143, so that one more retry is performed after an FTR -RELEASE command. (If the file is on hold because of an error that has not been corrected, the transfer fails again.)

If the Error Precludes Trying Again

If a user name or password is quoted incorrectly, the request is suspended by being put on hold. Either you or the system operator may:

- Delete the request on hold (using FTR -CANCEL) and then resubmit the request
- Correct it (using FTR -MODIFY) so that following a subsequent FTR -RELEASE command, the transfer succeeds
- Release it for another transfer attempt (using FTR -RELEASE)

OTHER OPTIONS

The FTR command permits other options that allow you to modify, abort, and otherwise control your requests. Full information on these options (as well as on the -HOLD and -RELEASE options) appears in Chapter 6, which describes each FTR command option.

6

An FTR Reference

INTRODUCTION

This chapter describes in two sections all of the options to the FTR command. Chapter 4 contains overview information on FTR. To learn which of these options you need in order to transfer files, see Chapter 5.

You request file transfers with submittal options. Those options are described in the first section of this chapter, OPTIONS FOR SUBMITTING REQUESTS. You manage, monitor, and cancel file transfer requests with management options. Management options are described in the second section, OPTIONS FOR MANAGING REQUESTS.

OPTIONS FOR SUBMITTING REQUESTS

FTR transfers files between local and remote sites. You can transfer files between users on a local site, but you cannot transfer files between two remote sites. FTR considers the Prime system you are logged into, and any remote disks that are added to that system, to be the local site. FTS requires that at least one of the sites be the local site. The following example shows the format of an FTR request.

```
FTR source-pathname [destination-pathname] [options]
```

Either source-pathname or destination-pathname may be a filename, if the file is in the current directory on the local site. If the file is in another directory, or if the directory is passworded, the complete

pathname (including the password, if needed), must be enclosed in single quotes. If a password is omitted or incorrectly specified, you receive this error message:

Passworded pathname must be fully qualified. (FTR)

Summary of Submittal Options

Here is a brief description of each file transfer request submittal option. Full descriptions follow.

Option	Abbreviation	Purpose
-COPY	-CPY	Transfers a copy of a file, not the original. (This is the default.)
-DELETE	-DL	Deletes the local source file after it has been transferred successfully. (The default is -NO_DELETE only if -NO_COPY is used; otherwise, the copy is deleted.)
-DEVICE	-DEV	Transfers a file to the default line printer (-DEVICE LP). (The default is null.)
-DSTN_FILE_TYPE	-DFT	Specifies either a SAM, DAM, SEGSAM, or SEGDAM destination file type. (The default is the source file type when sending, and null when fetching.)
-DSTN_NTFY	-DN	Sends messages to the destination user when a file transfer starts and ends. (The default is -NO_DSTN_NTFY.)
-DSTN_SITE	-DS	Specifies the site to which the file is to be transferred. (The default is the local site.)
-DSTN_USER	-DU	Specifies the file owner, or any user, at the destination site. (The default is null for the remote destination, login user-id for local destination.)

-HOLD	none	Holds the file transfer request in queue so that it is not initiated. (The default is to initiate a request.)
-LOG	none	Specifies the pathname of a log file. (The default is no log file.)
-MESSAGE_LEVEL	-MSGL	Specifies the level of detail entered in a log file. (The default is NORMAL, the minimum.)
-NAME	-NA	Specifies the name of the file transfer request. (The default is the source file name.)
-NO_COPY	-NCPY	Transfers the original source file, not a copy of it. (Sending a copy is the default).
-NO_DELETE	-NDL	Does not delete the local source file after it has been transferred successfully. (This is the default.)
-NO_DSTN_NTFY	-NDN	Does not send messages to the destination user when a file transfer starts and ends. (This is the default.)
-NO_SRC_NTFY	-NSN	Does not send messages to the source user when a file transfer starts and ends. (This is the default.)
-NO_QUERY	-NQ	Does not ask you questions when you submit or modify a request. (The default is to query you.)
-QUERY	none	Asks you questions when you submit or modify a request. (This is the default.)
-QUEUE	none	Specifies the name of the file transfer queue in which a request is to be placed. (The default is the configured remote site queue, or OPEN_SYSTEM, if that has been configured.)

-SRC_FILE_TYPE	-SFT	Specifies the source file type. Valid file types are SAM, DAM, SEGSAM, and SEGDAM. The default is the type of the local source file. Otherwise, the default is null.
-SRC_NTFY	-SN	Sends messages to the source user when a file transfer starts and ends. (The default is -NO_SRC_NTFY.)
-SRC_SITE	-SS	Specifies the site from which a file is to be transferred. (The default is the local site.)
-SRC_USER	-SU	Specifies the file owner, or other user, at the source site. (The default is null for the remote source; login user-id for the local source.)

Full Descriptions of Submittal Options

This section describes in detail each option that you might use when submitting a file transfer request. The section following this one describes how you manage those requests.

▶ -COPY

Abbreviation: -CPY

The -COPY option makes a copy of the file before transferring it. (This option has no effect on files requested from a remote source.) The copy that FTS makes and that is sent is deleted by the FTS server only after a successful transfer.

You can modify, rename, or delete the file after the request has been made. Changes do not affect the copy involved in the transfer. The -NO_COPY option causes the original file to be sent in whatever way it has been modified.

You should be aware that having FTS make a copy of the file will use up disk space. If the size of the local file being transferred is over 250K bytes, the following question appears.

```
Size of source file is xxxxxx bytes.  Ok to make a copy ?
```

Type Y, YES, or OK, to cause FTR to attempt to copy the file. Type N, NO, QUIT, Q, or press (CR) to transfer the original source file. Any other response causes the original source file to be transferred. If you specified -NO_COPY on the command line, the above question does not appear. You can use the -NO_QUERY option to suppress questions of this kind.

The specification of both -COPY and -NO_COPY options on the same command line causes an error. The -COPY option is the default.

▶ -DELETE

Abbreviation: -DL

The -DELETE option specifies that the original file, which you queued for transfer from the local site to a remote site, be deleted after it has been successfully transferred. This option applies to both -NO_COPY and to -COPY on local source files. This option does not affect files being transferred from a remote site to the local site.

The specification of both -DELETE and -NO_DELETE options on the same command line causes an error. The default is -NO_DELETE.

▶ -DEVICE LP

Abbreviation: -DEV

The -DEVICE option spools a file at the remote site's default line printer using the PRIMOS SPOOL command. You cannot specify any other line printer commands or SPOOL options with -DEVICE.

The following command line transfers the file MEMO from the local site to a remote site called MINOS and prints it on that system's line printer.

```
FTR MEMO -DEVICE LP -DS MINOS -DU CLARKE
```

The user banner is the name of the FTS server, and the file banner is either the name FTS_SPOOL_FILE or the name of the destination user, which is CLARKE in the above example.

Only sequential and direct access method (SAM and DAM) files can be transferred with this option. The default is null.

▶ -DSTN_FILE_TYPE { SAM | DAM | SEGSAM | SEGDAM }

Abbreviation: -DFT

The -DSTN_FILE_TYPE option specifies the file type of the destination file. Using this option, you can perform the conversions that you specify in the command line. You can convert SAM files to DAM files, and SEGSAM files to SEGDAM files (and vice versa for both pairs) between FTS Version 1 and FTS Version 2 systems. The default destination file type is the source file type when you send a file, and null when you fetch a file from another directory or remote site.

▶ -DSTN_NTFY

Abbreviation: -DN

The -DSTN_NTFY option causes the FTS server to send messages concerning the start and end of a file transfer to the logged-in destination user. Since FTS uses the PRIMOS MESSAGE facility to send these messages, this option cannot notify a user who is not logged in.

If you are fetching a file, the -LOG option is generally preferable to -DSTN_NTFY for learning the progress and results of a file transfer, since -LOG places transfer results in a file.

To send the -DSTN_NTFY message, FTS requires the name of the destination user. Use the -DSTN_USER option to supply this name. Omitting the destination user name causes an error, except when the destination site is the local site, in which case the submitting user-id is used.

The specification of both -DSTN_NTFY and -NO_DSTN_NTFY options on the same command line causes an error. The default for this option is -NO_DSTN_NTFY.

▶ -DSTN_SITE destination-site-name

Abbreviation: -DS

The -DSTN_SITE option specifies the destination-site-name to which the file is to be transferred. The length of the configured node name is limited to 32 characters. Because your System Administrator supplies the site address when configuring FTS, you only need to specify the site name in a file transfer request.

File transfers can take place between the local site and a remote site or between users at the local site, but not between two remote sites. Either the source site or destination site must be the local site. If no destination site is provided, the default is the local site. On occasion, you may want to transfer a file to a site that is not accessed frequently enough to be configured with FTGEN. In this case, must quote the open network address of the site in the destination site name. If you are using open network addressing and have to specify the numeric address along with the server name, the addressing may be up to 128 characters. The destination-site-name consists of three parts in the following format:

'address+server(password)'

The address may be either a site name (configured by CONFIG_NET, not by FTGEN), or a numeric address. The server is the name of the remote file transfer server process, which name is not necessarily the same as that of your local site's server. The password is the remote server's password. Single quotes must surround the entire destination site name. Two examples of open network addresses are as follows.

'SYS4+FTPS(THURSDAY)'

'311081800602+FTSRV(SECRET)'

In addition, you must specify the file transfer queue using the -QUEUE option, and the queue must be previously configured with FTGEN.

If you don't specify a queue, the default places the file transfer request on the OPEN_SYSTEM queue, if it has been configured at the submitting user's site. There is no default queue; one must be specified, such as OPEN_SYSTEM.

▶ -DSTN_USER destination-user-name

Abbreviation: -DU

The -DSTN_USER option specifies the destination-user-name. This identifies the user name or owner at the destination site of the file involved in the file transfer. The destination-user-name must conform to Prime user naming conventions and is limited to 32 characters. See Chapter 5 for examples of this option.

The default destination-user-name is your user-id, if the destination site for the file is the local site. For a destination site that is a remote site, the default is null.

▶ -HOLD

The -HOLD submittal option (HOLD is also a management option) causes the file transfer request you submit to be held on a file transfer queue. It insures that the request is not initiated until you say so. You free a held request for transfer with the -RELEASE management option, described in the section on OPTIONS FOR MANAGING REQUESTS.

Held requests can be modified before you release them. Requests that you hold can be released only by you or an operator.

This -HOLD option on submission serves the same purpose as the -HOLD options described under OPTIONS FOR MANAGING REQUESTS. The advantage of this -HOLD option over the other is that holding on submission ensures that the request is not initiated immediately, as it might be if there were no requests already outstanding on the request queue. The default is to initiate a request immediately.

▶ -LOG pathname

This option controls the automatic logging of file transfer request events, such as the start of a request submission, the name of the submitting user, the name of the file being transferred, the result of the transfer, and its termination. The -LOG option is recommended as preferable to the -SRC_NTFY and -DSTN_NTFY options for tracking the progress and results of a file transfer, since you must specify a file that you can refer to later.

The log file specified in the pathname is a text file, and is similar to a command output file. Chapter 4 has an example of log file entries. Whenever an event occurs to a file transfer request, FTS records it in this file. You can use the -MESSAGE_LEVEL option to specify the level of detail in the log file (NORMAL is the usual detail level).

If the specified log file already exists, new log entries are appended to it. You may want to periodically remove the old entries that build up in a log file that is in continual use.

If the log file specified does not exist, it is created. A log file created by the FTS will have its read/write lock set to UPDT (see the RWLOCK command in the Prime User's Guide), to allow updating by FTS while you are reading the file. The File Transfer Manager must have DALURWX rights to your directory in order to write the results of a file transfer request to the log file.

The length of pathname is limited to 128 characters. If pathname includes a password, it must be a full pathname enclosed in single quotes. The default is to not generate a log file.

```
▶ -MESSAGE_LEVEL { NORMAL     }
                 { DETAILED   }
                 { STATISTICS }
                 { TRACE      }
```

Abbreviation: -MSGL

The -MESSAGE_LEVEL option specifies the amount of information that is entered in the log file, which you specify with the -LOG option. In order to use -MESSAGE_LEVEL, you must also use -LOG, or an error occurs.

The following message levels can be specified.

Message Level	Abbreviation	Function
1 NORMAL	NRM	Enters minimum of detail in the log. This is the default.
2 DETAILED	DET	Logs all events.
3 STATISTICS	STAT	DETAILED information is logged, with statistics.
4 TRACE	TRC	DETAILED, STATISTICS, and TRACE information is logged.

For example,

```
FTR MEMO <MEMDSK>RANDUM -DSTN_SITE BIRCH -DSTN_USER CLARKE
-LOG RECORD -MESSAGE_LEVEL DETAILED
```

specifies a detailed log of the transfer called MEMO to a file called RANDUM on the remote system BIRCH, in the log file RECORD at the local site.

Each log message includes the number of the server link to which the message relates and the numerical value of the message level at which the message is logged. For example,

```
10.17.24: [1.2] Remote file is <TSTDSK>RECEIVE>FILE1
```

refers to the request that was active on server link 1, and has been logged at the DETAILED log level. If you do not specify -MESSAGE_LEVEL when you submit a request, the default message level is NORMAL.

▶ -NAME request-name

Abbreviation: -NA

You can specify the name of the request in request-name. This name uniquely identifies the request, providing you do not have other requests with the same name.

The default name for a request is the name of the file being transferred. For example, the default name for a source pathname of "FOO>BAR>XYZ" would be "XYZ". Any name you use should conform to Prime file naming standards and is limited to 32 characters.

▶ -NO_COPY

Abbreviation: -NCPY

The -NO_COPY option causes the current copy of a file, rather than the latest copy of it, to be transferred.

You should be aware that any changes you make to this file between the time you make the file transfer request and the time when the file is transferred are included in the transfer.

The -COPY option results in a copy of the original file being taken and the copy being used in the file transfer. The specification of both -NO_COPY and -COPY options on the same command line is not allowed and causes an error. The default is to transfer a copy of the file (-COPY).

▶ -NO_DELETE

Abbreviation: -NDL

The -NO_DELETE option specifies that the local source file, which is to be transferred from the local site to a remote site, not be deleted after it has been successfully transferred. This default applies to both -COPY and to -NO_COPY transfers. The specification of both -NO_DELETE and -DELETE on the same command line causes an error.

▶ -NO_DSTN_NTFY

Abbreviation: -NDN

The -NO_DSTN_NTFY option does not notify the destination user of the start and end of a file transfer. The specification of both -DSTN_NTFY and -NO_DSTN_NTFY on the same command line is not allowed and causes an error. This option is the default.

▶ -NO_QUERY

Abbreviation: -NQ

The -NO_QUERY option ensures that you are not queried when submitting or modifying a file transfer request. This option is particularly useful for running FTS from a CPL file. See the -COPY option for an example of a user query.

The specification of both -NO_QUERY and -QUERY options on the same command line is not allowed and causes an error. The default is that FTS queries you, when necessary.

▶ -NO_SRC_NTFY

Abbreviation: -NSN

The -NO_SRC_NTFY option ensures that no messages are sent by the file transfer server to the source user to indicate the start and end of a file transfer.

The specification of both -SRC_NTFY and NO_SRC_NTFY on the same command line causes an error. This option is the default.

▶ -QUERY

The -QUERY option (there is no abbreviation for -QUERY) ensures that you are asked to confirm a submitted or modified request. (See the description of the -COPY option for an example of a user query.)

The -NO_QUERY option suppresses such user queries for that request only. The specification of both -QUERY and -NO_QUERY options on the same command line is not allowed, and causes an error. This option is the default.

▶ -QUEUE queue-name

The -QUEUE option specifies the queue-name in which the file transfer request is to be placed. The length of the queue name is limited to 32 characters.

Normally, this option is used only when sending files to a remote site that has not been configured by FTGEN. In this case, the option must be used and must specify a queue that has been configured with FTGEN. The default queue is OPEN_SYSTEM, if that has been configured. (See your System Administrator for a list of file transfer queues within your network.)

The -QUEUE option is seldom used for transfers to configured sites. These requests are automatically placed on the queue configured for the remote site by the System Administrator. If the -QUEUE option is used in this situation, it overrides the default setting.

▶ -SRC_FILE_TYPE { SAM | DAM | SEGSAM | SEGDAM }

Abbreviation: -SFT

The -SRC_FILE_TYPE option specifies the source-file-type. For fetching files, you can use this option to specify the type of a remote source file. Before transfer commences, the FTS server determines whether or not the remote source file type equals the one specified. The transfer is rejected if they do not match. The default for local source files is the type of the source file; otherwise, it is null.

▶ -SRC_NTFY

Abbreviation: -SN

The -SRC_NTFY option causes the FTS server to send messages to you concerning the start and end of a file transfer. Since FTS uses the PRIMOS MESSAGE facility to send these messages, this option cannot notify you when you are not logged in.

If you are fetching a file, the -LOG option is generally preferable to -SRC_NTFY for learning the progress and results of a file transfer, since -LOG places transfer results in a file.

To send the -SRC_NTFY message, include the -SRC_USER option on the command line. You don't have to specify your name, since FTS uses the logged-in user-id as the default.

The specification of both -SRC_NTFY and -NO_SRC_NTFY options on the same command line causes an error. The default for this option is -NO_SRC_NTFY.

▶ -SRC_SITE source-site-name

Abbreviation: -SS

The -SRC_SITE option specifies the source-site-name from which the file is being transferred. The length of the configured node name is limited to 32 characters. When your System Administrator has configured FTS, you need only to specify the site name in a file transfer request. FTS refers to its configuration for all site addresses.

File transfers can take place between the local site and a remote site or between users at the local site, but not between two remote sites. Either the source site or destination site must be the local site. You don't have to specify the local site, since FTS uses this as the default. FTS gets the required information on sites from its configuration.

On occasion, you may want to do a file transfer from a site which is not accessed frequently enough to have been configured. In this case, you must quote the open network address of the site as the source site name. Open network addressing for -SRC_SITE is identical to that for -DSTN_SITE. See the description of -DSTN_SITE for details on the format of open network addressing.

▶ -SRC_USER source-user-name

Abbreviation: -SU

This option identifies the username or owner at the source site of the file involved in the file transfer. The source-user-name must conform to Prime user naming conventions and is limited to 32 characters.

The default is your user-id if the source site for the file is the local site. The default is null when the source site is a remote site.

OPTIONS FOR MANAGING REQUESTS

Use the following options to manage your own submitted file transfer requests. If you are logged in as SYSTEM, you gain FTS operator privileges that allow you to manage file transfer requests submitted by any user on the system.

Note

FTR commands that you use while logged in as SYSTEM affect all FTR user-submitted requests.

The options are used in the following format.

```
FTR option [ request-name   ]
           [ request-number ]
```

Summary of Management Options

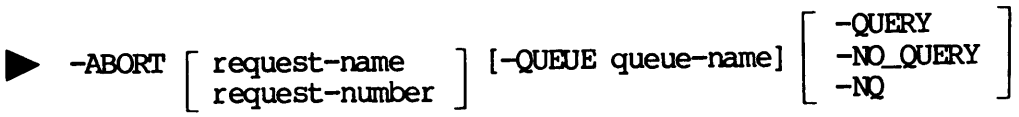
FTR management options do not use abbreviations. The following table shows a brief description of each FTR management option.

Option	Meaning
-ABORT	Aborts one or more file transfer requests.
-CANCEL	Cancels one or more file transfer requests.
-DISPLAY	Prints the details of one or more requests.
-HELP	Displays help information on FTR.
-HOLD	Delays one or more file transfers until a user or operator releases the request using the -RELEASE option.
-MODIFY	Modifies the characteristics of one or more requests.
-RELEASE	Releases one or more held requests.
-STATUS	Displays the status of one or more requests.
-STATUS_ALL	Prints the status of all file transfer requests.

Full Descriptions of Management Options

Each FTS site has a default file transfer server and file transfer queue. Requests are placed on that queue until the server associated with the queue transfers the queued requests.

Usually, all sites have the same queue associated with them. However, depending on your local FTS configuration, different sites (or groups of sites) may have different queues associated with them. In such cases, the -QUEUE option allows you to specify which queue you want a request to go to. See you System Administrator for a list of file transfer sites and their associated transfer queues.



The -ABORT management option aborts one or more requests, even if the transfers are already in progress, and puts them on hold. See the -HOLD option below for more information.

Requests that are not in an eligible state to be aborted (for example, those already aborting), do not cause an error. Instead, FTR finds and aborts all eligible requests.

The -QUEUE option can be used to restrict the requests that are aborted to those found in queue-name.

If you specify a request, the aborted request names and numbers are displayed.

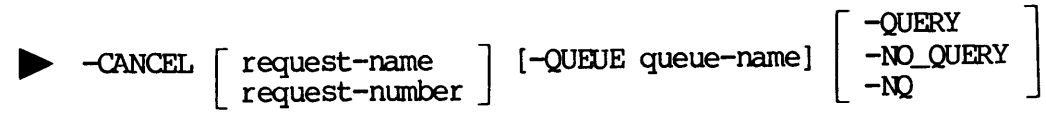
If you don't specify a particular request, you see the following question.

O.K. to abort all requests?

You can suppress the question with the '-NO_QUERY' option (abbreviated -NQ). If you don't want to abort all of your requests, type 'N', 'NO', 'QUIT', 'Q', or press (CR). To abort all requests, type 'Y', 'YES', or 'OK'. Any other response indicates a negative response.

Aborted request names and numbers are not displayed. Instead, the following statement appears.

All eligible requests aborted.



The -CANCEL management option deletes one or more requests from a file transfer request queue. If the transfer is already in progress, or aborting, then the request is not deleted.

Requests that are not in an eligible state to be canceled (for example, those already in progress or aborting), do not cause an error. Instead, FTR finds and cancels all eligible requests. Canceled request names are not displayed. Instead, the following statement appears.

All eligible requests cancelled.

If you specify a request-name or a request-number, then that specific request is canceled on any configured file transfer queue. The request name and number are displayed.

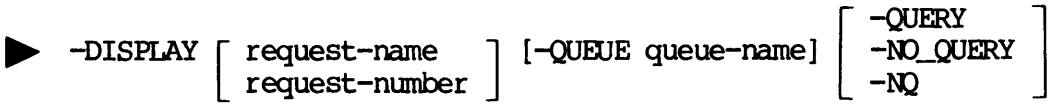
If you don't specify a request name or number, then all requests you submitted, on any configured queue, are canceled, and the message above is displayed.

You can use the -QUEUE option to restrict the requests that are canceled to only those requests found in queue-name. If you specify a request, the canceled request names and numbers are displayed.

If you don't specify a particular request, you see the following question.

```
O.K. to cancel all requests ?
```

You can suppress the question with the '-NO_QUERY' option (abbreviated NQ). If you don't want to cancel all of your requests, type 'N', 'NO', 'QUIT', 'Q', or press (CR). To cancel all requests, type 'Y', 'YES', or 'OK'. Any other response indicates a negative response.



The -DISPLAY management option shows detailed information about requests. Information includes all that is given by the -STATUS option and all that is included in the request.

For all formats of the FTR -DISPLAY command, passwords contained in the source file pathname, destination file pathname, or log file pathname are only displayed for requests that you own. For other requests, passwords are removed from these pathnames. The following information is included in the display output.

- Source and destination sites
- Whether the file is to be deleted after the transfer
- Request log file name
- Source and destination pathnames
- The queue on which the request is residing

If you specify a request-name or a request-number, then that specific request, on any configured file transfer queue, is displayed.

If you don't specify a request name or number, all requests that you submitted, on any configured queue, are displayed.

The -QUEUE option can be used to restrict the requests that are displayed to only those requests in queue-name.

If more than one request is output, you are asked between requests if you want to continue output. To suppress the display of this prompt, use the -NO_QUERY option (abbreviated -NQ).

▶ -HELP [subject]

The -HELP management option displays help information. To obtain a list of subjects on which help is available, type either

```
FTR -HELP SUBJECTS
```

or simply,

```
FTR -HELP
```

To get help on using FTR command options, type either

```
FTR -HELP USAGE
```

or simply,

```
FTR
```

▶ -HOLD [request-name | request-number] [-QUEUE queue-name] [-QUERY | -NO_QUERY | -NQ]

The -HOLD management option holds one or more specified requests. The requests are not initiated until they are released. (See the -RELEASE option.) If the requests are in a state other than waiting, the command has no effect.

Requests that are not in an eligible state to be held (such as those in progress, already held, or aborting) won't cause an error message to appear. Held request names are not displayed. Instead, the following statement appears.

```
All eligible requests held.
```

If you specify a request-name or a request-number, that specific request is held on any configured file transfer queue. The request name and number are displayed.

If you don't specify a request name or number, all requests you submitted, on any configured queue, are held, and the message shown above is displayed.

The -QUEUE option can be used to hold only those requests in queue-name.

When it displays the requests that have been held, FTR includes a prompt for the display to be continued (--More--) after every 23 lines. Use the -QUERY option to suppress this prompt.

▶ -MODIFY [request-name | request-number] [legal-options]

The -MODIFY management option modifies the characteristics of a submitted file transfer request prior to initiation.

Requests that are not in an eligible state to be modified (for example, those in progress or aborting) do not cause an error message to appear. Instead, FTR modifies all eligible requests that are found. Held requests are not retried until they have been released. Once the requests have been modified, the following message appears.

All eligible requests modified.

If you do specify a request-name or a request-number, all requests having that name or number, on any configured queue that belongs to you, are modified. The modified request names and numbers are displayed.

If you don't specify a request name or number, all requests belonging to you are modified in all configured queues. The message shown above is displayed.

You can modify any transfer option, with the following exceptions:

-QUEUE	-NO_COPY	-COPY
-DSTN_SITE	-SRC_SITE	-HOLD
-DSTN_FILE_TYPE	-SRC_FILE_TYPE	

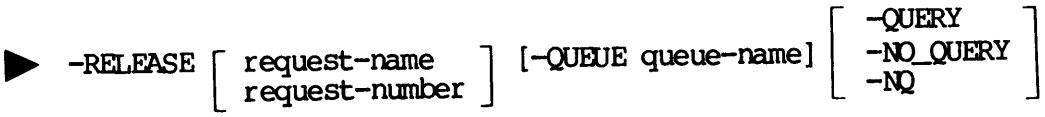
For details of the available options and their syntax, refer to the Request Submittal section earlier in this chapter.

Modifying the characteristics of a request is similar to canceling a request and resubmitting it. The -MODIFY option is different in that the request remains in the same position in the queue. Therefore, modifying a request generally does not affect the time when the request is initiated. However, canceling and resubmitting a request would probably delay the initiation time of the transfer.

If you don't specify any legal options, the requests will have the following items modified.

- Date and time of last retry is set to zero
- Number of retries is set to zero

If the requests are waiting, this makes them eligible for immediate retry by the file transfer server without waiting until the next 30-minute retry period has expired. This can be useful when a previously inoperative site becomes operational and the you would like your pending request to be retried as soon as possible.



The -RELEASE management option initiates a file transfer request that was previously held by the -HOLD Request Management option, the -HOLD Request Submittal option, or by FTS.

Requests that are not in an eligible state to be released, such as those waiting, transferring, or aborting, do not cause an error. FTR finds and releases all eligible requests. Released request names are not displayed. Instead, the following statement appears.

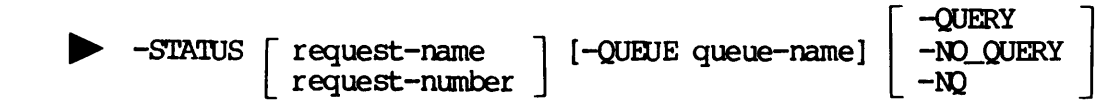
```
All eligible requests released.
```

If you specify a request-name or a request-number, that specific request is released on any configured file transfer queue. The name and number is displayed.

If you don't specify a request name or number, then all the requests that you submitted, on any configured queue, are released. The message shown above is displayed.

You can use the -QUEUE option to release only those requests in queue-name. If you specified a request, the released request names and numbers are displayed. If you didn't specify a particular request, all requests are released.

When it displays the requests that have been released, FTR includes a prompt for the display to be continued (—More—) after every 23 lines. Use the -QUERY option to suppress this prompt.



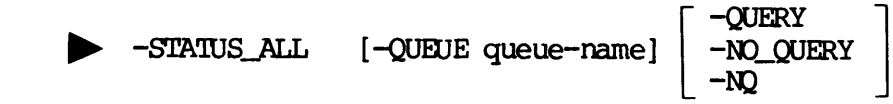
The -STATUS management option displays information about the current status of the request. For operators logged in under the user-id SYSTEM, it displays information on all users. The following information is returned by -STATUS for each request.

- Date and time the request was queued
- User name of the submitting user
- Name and number of the request
- The queue on which the request resides
- The current status of the request

If you specify a request-name or a request-number, that specific request, on any configured file transfer queues, is shown.

If you don't specify a request name or number, then all requests you submitted, on any configured queue, are shown.

You can use the -QUEUE option to show only those requests in queue-name. If more than a page (22 lines) is output, you are asked at the end of each page whether you want to continue or not. To suppress this prompt, use the -NO_QUERY option (abbreviated -NQ).



The -STATUS_ALL management option returns information about all the currently queued file transfer requests on all the configured transfer queues. You can also see where your requests are in the queue.

Note

This command provides the same information as the STATUS option, with the exception that it provides information on all user requests, not just your own.

The information that is returned by -STATUS_ALL for each request is identical to that in -STATUS.

- Date and time the request was queued
- User name of the submitting user
- Name and number of the request
- The queue on which the request resides
- The current status of the request

You can use the -QUEUE option to restrict output to only those requests in queue-name. If -QUEUE is not used, the status of all requests, on all configured queues, is printed.

If more than a page (22 lines) is to be output, you are prompted at the end of each page for output to continue or not. To suppress this prompt, use the -NO_QUERY option (abbreviated -NQ).

PART IV

NETLINK

7

Introduction to NETLINK

INTRODUCTION

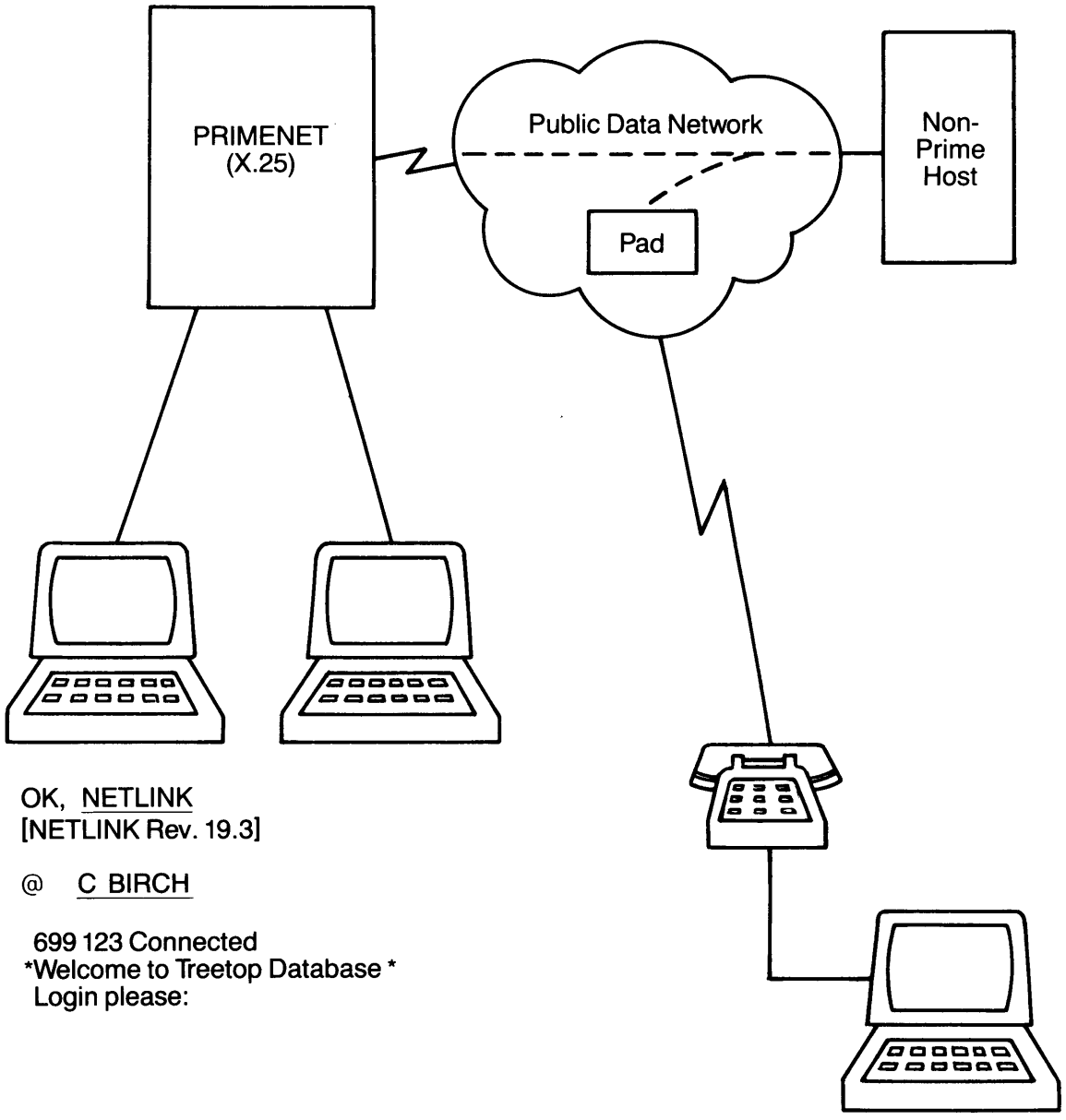
The NETLINK utility provides you with alternative methods of remote login and file transfer for use in certain circumstances, particularly between systems linked through PDNs. This chapter introduces

- NETLINK's modes of operation
- Network addresses
- NETLINK file transfers

Chapter 8 describes NETLINK commands that let you connect, transfer and receive files, disconnect, call, quit, pause, and continue from a remote network node. Chapter 9 provides a full description of all NETLINK commands, options, parameters, and error messages.

WHAT IS NETLINK?

NETLINK is an interactive utility for making up to six remote login connections in any PRIMENET environment and in non-Prime environments, such as Public Data Networks (PDN). For example, if your Prime system is connected to TELENET, you can use NETLINK to connect to any other TELENET system, whether or not that remote system is known to PRIMENET. Figure 7-1 shows a diagram of NETLINK.



NETLINK
Figure 7-1

In addition, NETLINK supports the CCITT 1980 Recommendations X.3 and X.29, allowing you to configure certain parameters (such as what happens if the BREAK key is pressed) and tailor NETLINK to the needs of the remote system. For example, you can slow down the data transmission rate to allow a linked remote sytem to keep up with your local Prime computer.

The only requirements for using NETLINK are:

- Knowledge of the remote node's id or X.25 address
- A user-id on the remote node
- Knowledge of any other user-entry validations

NETLINK'S MODES OF OPERATION

NETLINK is always in one of two modes, command mode or data transmission mode. Command mode allows you to interact with NETLINK, giving it commands to

- Establish or clear connections
- Modify NETLINK or PDN parameters associated with a particular connection
- Change the operation of NETLINK
- Switch between open connections

Data transmission mode allows you to interact with a remote host to which you have established a NETLINK connection. All characters that you type on your keyboard, except for certain escape sequences, are passed directly to the remote host. This allows you to login normally and to perform other operations.

Command Mode

NETLINK indicates that it is in command mode by displaying a prompt character. Normally it displays @, but you can change this character with the PROMPT command. (Chapter 8 shows an example of the PROMPT command.) A different prompt would be helpful if you are connecting to several remote hosts or using a subsystem on the remote host that uses the @ prompt.

Data Transmission Mode

NETLINK enters data transmission mode when a new connection is initiated, or when a suspended connection is reactivated. In this mode, all characters you type on your keyboard (with two exceptions) are transmitted to the remote host of your active circuit.

The exceptions are:

- The NETLINK escape character (normally @)
- The BREAK key

Pressing RETURN, typing the escape character (@), and pressing RETURN again returns you to NETLINK command mode. To return to data transmission mode, use the CONTINUE command.

To send a single escape character as data at the beginning of a line to the remote host, you must double it. That is, to send @ alone you must type @@.

The action of the BREAK key depends upon the value of X.3 parameter number 7. See the descriptions of the SET and PAR commands in Chapter 9 and in Appendix B. The default is to send a break signal to the remote host; this leaves NETLINK in data transmission mode.

NETWORK ADDRESSES

To connect to a remote host computer, you must know its name or its numeric address. Machines can have many addresses, but only one name. The address is either a name (such as BEECH) or a set of numbers (such as 413 788). To learn the addresses (name or number) of computers you can access, ask your System Administrator. The PRIMOS command STATUS NETWORK lists the configured systems in your network. For specifying an address, see Chapter 8.

Because NETLINK can operate over any PRIMENET connection, not just through PDNs, you can use it to access any Prime system in a network as well as using Remote Login.

NETLINK FILE TRANSFERS

You can use NETLINK's FILE and OUTFILE commands to copy textual data either from a local to a remote host or from a remote to a local host. Examples of these commands can be found in Chapter 8; the commands are described in Chapter 9.

If the remote system is not a Prime system, NETLINK can be used to transfer text files. Prime's File Transfer Service (FTS) provides an efficient way of transmitting data from one Prime system to another. FTS or the PRIMOS COPY command, however, are preferred methods for transferring data to and from other Prime hosts. (See Chapter 4 for a comparison of FTS and COPY.)

NETLINK may be the only way to exchange data over an X.25 network between Prime and non-Prime systems. Since NETLINK may be talking to a non-Prime host, no protocol can be used, so data transparency cannot be provided. Therefore, only text can be transferred.

8

Using NETLINK

INTRODUCTION

This chapter shows you examples of the following commonly-used NETLINK command procedures.

- Connecting to a remote system
- Logging into a remote system
- Making multiple connections
- Changing the NETLINK prompt
- Transferring a file with FILE and OUTFILE
- Running NETLINK from a command input file
- Showing the profile of a circuit
- Displaying X.3 parameters
- Debugging with NETLINK

To use NETLINK's capabilities fully, you should have some general knowledge of X.3, X.29, and Public Data Network (PDN) operation. (This knowledge is not necessary for most common uses of NETLINK.) For special applications, you may need to refer to documentation from the CCITT or from your PDN. Chapter 9 has a complete description of all NETLINK commands.

INVOKING NETLINK

After you type the NETLINK command, you will see the command mode prompt (@), indicating that you can issue NETLINK commands. The following is an example of invoking NETLINK.

```
OK, NETLINK
[NETLINK Rev. 19.3]

@
```

NETLINK ordinarily starts up in command mode. While in command mode, you can type HELP for a brief description of the NETLINK commands. Once you have connected to another system, you can type STATUS to request information about active circuits.

You can temporarily escape from command mode and return to PRIMOS by typing the PAUSE command. You can then use any PRIMOS internal command. Typing an S returns you to command mode. The QUIT command ends a NETLINK session and returns you to PRIMOS.

MAKING A CONNECTION

Use the C or NC command to connect to a remote host. The commands have the following format.

```
C address [options]

NC address [options]
```

Use the C command for most connections. For a reverse charges or collect call (C), any network communications charges will be billed to the remote host, which presumably passes those charges on to the logged-in user.

Many hosts, especially hosts in PDNs, do not allow collect calls. Use the NC command to hosts that do not permit collect calls. Use NC also for intra-network or trans-network connections. For an NC call, any charges are billed to the originating site.

ADDRESSING ANOTHER SYSTEM

The examples below show how to address another system through NETLINK. The first method uses PRIMENET-configured names. In this method, an address is a nodename up to six characters long. Your System Administrator can tell you the addresses and names that are configured in your system's network. In configuring PRIMENET, the System Administrator assigns names to the addresses of remote systems. The second method, used for systems that do not have a name associated with it, uses the systems' PDN addresses.

PRIMENET-configured Name Addressing

This method of addressing uses the configured name of a remote system. NETLINK automatically looks up and uses the correct address. For example,

```
@ C MKTG.5
```

establishes a connection to the remote computer MKTG.5. Configured systems do not have to be Prime systems. Your System Administrator could configure other machines as PDN hosts. In this case, users would only need to specify C OTHER, for example, to connect to a remote non-Prime system.

Public Data Network (PDN) Addressing

Open or PDN addressing can be used for connections over TELENET and other PDNs. It uses the following format.

```
dnic:area number.ss
```

dnic — is an optional four-digit Data Network Identification Code. The default is TELENET's code, 3110. See the DNIC command in Chapter 9.

area — is the three-digit code.

number — is the address of one to five digits. Leading zeros are not required. This is an arbitrary number assigned by the PDN to represent your computer. For example, 117 74 could be the 74th computer in a network of 117 computers.

ss — is the optional two-digit subaddress.

Below are two examples of TELENET addresses. The second example shows explicit use of the dnic, which is ususally required by other connections. Neither example uses the 2-digit subaddress.

```
@ C 619 73

@ C 4556:706 189
```

International Addressing

International connections require their own particular format, starting with the appropriate national address code for dnic.

Literal Addressing

NETLINK supports the full X.121 14-digit standard address format. The address can be specified as a string of 12 to 14 digits, preceded by a colon. The two rightmost digits, 13 and 14, are the optional subaddress. An example (without a subaddress) could be expressed as follows.

```
@ C :455660500073
```

Connect Packet Options

You can use NETLINK to connect your Prime computer to any X.25-compatible host computer. For connections to non-Prime hosts or through international gateways, you might need to specify connect packet options, which are shown in the description of the C command in Chapter 9.

DIRECT REMOTE LOGIN

The simplest use of NETLINK is for remote login. NETLINK allows you to maintain up to six simultaneous remote login connections and to switch your terminal between these connections as you desire. You can connect directly to a remote system, and skip the NETLINK command mode by using the -TO option, as follows.

```
OK, NETLINK -TO BIRCH
BIRCH Connected
Save the Trees, Inc.      /* This example shows a PRIMOS login
Central Computer          /* command line, which is valid only
                          /* between Prime systems.

Enter access code please:
```

Note

The -TO option is especially useful in abbreviations, made by using the ABBREV command of PRIMOS. For an example, refer to the entry for the TO command in Chapter 9. (All NETLINK command line options are also NETLINK commands.)

MULTIPLE CONNECTIONS

You can have up to six separate NETLINK connections active at once. After you have made your first connection as described above, you make your subsequent connections differently. First, you must return to command mode by typing an @ alone on a line (that is, at the beginning of a line, type @ and press (CR)). Then issue another C or NC command.

You can switch between active circuits with the CONTINUE command (abbreviated CO), or with the SWITCH command (abbreviated SW). Use the D (disconnect) command to disconnect circuits. You remain in NETLINK command mode and the other active circuits are intact. The following example shows a system-to-system connection, and the SWITCH command.

```
OK, NETLINK
[NETLINK  Rev. 19.3]

@ C ELM

ELM Connected
PRIMENET 19.3 ELM
LOGIN SMITH
Password? ANTARES

OK, STATUS NETWORK

     Node          State
ELM                ****
PINE                Up
BEECH               Up
CHERRY              Up
BIRCH              Down
MAPLE               Up
LINDEN             Down
ASH                 Up
LAUREL              Up
CACTUS              Up
FIR                 Up
TREE                Up
```

```
OK, @

@ C ASH

ASH Connected
PRIMENET 19.3 ASH
LOGIN SMITH
Password? SIRIUS

OK, @          /* You have to escape to NETLINK command */
                    /* mode on the originating site      */
@ STATUS

  Circuit #       Address              Time    Pkt in     Pkt out
  ---------   --------------------     ----   --------   --------
      1       ELM                      0:00          2          1
-->   2       ASH                      0:00          4          3

@ SW 1

@ STATUS

  Circuit #       Address              Time    Pkt in     Pkt out
  ---------   --------------------     ----   --------   --------
-->   1       ELM                      0:00          2          1
      2       ASH                      0:00          4          3
@ QUIT

ELM                 Disconnected
ASH                 Disconnected
OK, LO
```

EXAMPLE OF THE PROMPT COMMAND

This command changes the NETLINK command prompt (@) to whatever text string you want. You may want to change the prompt if you are making multiple connections to systems through NETLINK. The following example makes a connection from system LINDEN to ELM, then returns to LINDEN.

```
OK, NETLINK
[NETLINK  Rev. 19.3]

@ PROMPT LINDEN>

LINDEN> C ELM

ELM Connected
PRIMENET 19.3 ELM
LOGIN SMITH
Password? TAXES
```

```
SMITH (user 56) logged in Wednesday, 03 Aug 83 00:41:32.
Welcome to PRIMOS version 19.3.
Last login Wednesday, 03 Aug 83 00:40:24.


Enter validation code: LAYER

OK, NETLINK
[NETLINK  Rev. 19.3]

@ PROMPT ELM>

ELM> @

@ QUIT

ELM                         Disconnected

OK,
```

EXAMPLES OF THE FILE AND OUTFILE COMMANDS

The following examples show how you would use the FILE and OUTFILE commands to transfer a file to ED or to an equivalent editor on a remote system. The file you specify is used and forwarded only after you type CO to transfer data to the remote system.

File Transfer Capabilities

You can successfully transfer files consisting of printing characters (including end-of-line CR or LF characters). Files can be transferred from any site that can display a file as does the PRIMOS command SLIST, and to any site that can accept typed input into a file as does the PRIMOS command ED.

Precautions

The local or remote site that receives the transmitted file will most likely have an erase character and a kill character. The default erase and kill are the double quote (") and the question mark (?). (These are configurable at PRIMOS cold start.) The file in NETLINK's FILE command must not contain these characters. NETLINK's OUTFILE command, used to receive files, does not process erase and kill characters.

If the site transmitting the file inserts padding characters, such as NULs or DELs after CR, the characters appear in the received file.

Local-to-remote File Transfers

The FILE command sends text files to a connected remote host. This is performed in the following manner.

1. Make a connection to the remote host and log in.
2. Start ED (or an equivalent) and go into INPUT (or an equivalent) mode.
3. Escape from the remote host by typing an "@ (CR)" sequence (that is, @ alone on a line). NETLINK will return to command mode and respond with an @ prompt.
4. Enter the FILE command, the file pathname, and any options. (Refer to the FILE command in Chapter 9.)
5. Return to data transfer mode with a CONTINUE command.
6. When the transfer is complete, the message "End of file. filename" appears on your terminal. In ED, save the file on the remote system.
7. Exit from ED (or equivalent) on remote host.
8. Return to NETLINK Command Mode and enter a CLOSE FILE command.

The following example illustrates a local-to-remote file transfer between Prime systems. This example begins after you already have logged into the remote host.

```
OK, ED
INPUT

@

@ FILE PINE.TRL -PAD

@ CONTINUE

This is a test of the NETLINK file transfer facility.

                              /* The above is text in PINE.TRL
                              /* that is displayed. */

End of File. PINE.TRL         /* Press (CR) here. */

EDIT

FILE PINE.TRL
OK, @

@ CLOSE FILE
```

Remote-to-local File Transfers

The OUTFILE command copies output from a connected remote host to a disk on your local host. Use it in the following manner to transfer files.

1. Make a connection and log in to the remote host.
2. Escape to the local host with an "@ (CR)" sequence. NETLINK returns to command mode and displays an @ prompt.
3. Enter an OUTFILE command with the pathname and any options.

> **Note**
>
> If a file of that name already exists, that file is overwritten, so make sure that the filename you type in with the OUTFILE command is unique.

4. Return to the remote host with a CONTINUE command.
5. Issue an SLIST (or equivalent) command on the remote host to list the file.
6. Escape to the local host and do a CLOSE OUTFILE command.

The following example illustrates a remote-to-local file transfer between Prime systems. In this example, we are logged into the remote host, and have entered a (CR) to return to command mode.

```
@ OUTFILE PINE.TRL

@ CONTINUE

OK, SLIST PINE.TRL          /* Text of PINE.TRL appears */
                                /* on your terminal. */

OK, @
@ CLOSE OUTFILE
```

RUNNING NETLINK FROM A COMMAND INPUT FILE

If you want to use NETLINK from a command input (COMI) file to log in to a remote system, you need to create two files. One file should contain login information such as user-id, password, and any additional validation information. The other file is a command input file containing NETLINK commands that make the connection, open the file containing login information, and quit from the connection at the end of a session. Examples of the two files appear below.

FILE CONTAINING LOGIN INFORMATION

```
OK, SLIST BIRCH.LOGIN

LOGIN SMITH                 /* Your user id */
DANDELION                   /* Your password */
FLOWER                      /* Validation information */
```

COMI FILE

```
OK, SLIST BIRCH.COMI
NETLINK                     /* Invoke NETLINK.  */
CALL BIRCH                  /* Connect to system BIRCH.  */
                            /* Treat input from terminal or file */
                            /* as NETLINK commands.  */
FILE BIRCH.LOGIN -LINE -NTTY
                            /* Read login information from file */
                            /* BIRCH.LOGIN;  option -NTTY suppresses */
                            /* output of file to the screen.  */
CONTINUE                    /* Complete connection to BIRCH;  */
                            /* leave NETLINK command mode.  */
QUIT                        /* Disconnect from BIRCH when user */
                            /* types @.  */
CO -E                       /* End COMI file.  */
```

Note

You should place an ACL on your login file so that your password remains secure.

The following example shows the execution of the COMI file, BIRCH.COMI. You type in only the first command line; the COMI file does the rest of the work.

```
OK, COMI BIRCH.COMI
[NETLINK  Rev. 19.3]

@ CALL BIRCH


@ FILE BIRCH.LOGIN -LINE -NTTY

@ CONTINUE

BIRCH Connected
PRIMENET 19.3
```

```
End of file. ENB.LOGIN

Password?

SMITH (user 64) logged in Tuesday, 07 Feb 84 13:33:12.
Welcome to PRIMOS version 19.3
Last login Monday, 06 Feb 84 08:45:20.


Enter validation code:

OK, ED
...
etc,

OK, @

@ QUIT

BIRCH Disconnected
OK, CO -E
```

EXAMPLE OF THE PROFILE COMMAND

This command shows the profile of the current circuit. The parameters shown include debug mode, terminal speed, and node addresses. You can set a PROFILE for all or just one connection.

```
OK, NETLINK
[NETLINK  Rev. 19.3]

@ C ELM

ELM Connected
PRIMENET 19.3  ELM
@

@ PROFILE

Operational Parameters

  Debug: Off
  Polling time:    0.5 Secs.
  Escape character: '@'
  Terminal type:   UNKNOWN
  Terminal speed:  1200 bps

Connect parameters

  Dnic:    4556
  Address: ELM
  Port:    None
```

```
  Facilities:  Reverse Charging
  Protocol ID: (in decimal)   1   0   3   0
  User Data:   USERNAME

@ PROFILE DEFAULTS

Operational Parameters

  Debug: Off
  Polling time:    0.5 Secs.
  Escape character: '@'
  Terminal type:   UNKNOWN
  Terminal speed:  1200 bps

Connect parameters

  Dnic:    4556
  Address: None
  Port:    None
  Facilities:  Reverse Charging
  Protocol ID: (in decimal)   1   0   3   0
  User Data:   USERNAME

@ C ASH

ASH Connected
PRIMENET 19.3. ASH
@

@ PROFILE

Operational Parameters

  Debug: Off
  Polling time:    0.5 Secs.
  Escape character: '@'
  Terminal type:   UNKNOWN
  Terminal speed:  1200 bps

Connect parameters

  Dnic:    4556
  Address: ASH
  Port:    None
  Facilities:  Reverse Charging
  Protocol ID: (in decimal)   1   0   3   0
  User Data:   USERNAME

@ DATA PASSWORD       /*  Changes the User Data field
                          in the next connect. */
@ C PINE

PINE Connected
PRIMENET 19.3  PINE
```

```
@

@ PROFILE

Operational Parameters

  Debug: Off
  Polling time:    0.5 Secs.
  Escape character: '@'
  Terminal type:   UNKNOWN
  Terminal speed:  1200 bps

Connect parameters

  Dnic:    4556
  Address: PINE
  Port:    None
  Facilities:  Reverse Charging
  Protocol ID: (in decimal)   1   0   3   0
  User Data:   PASSWORD                        /* User Data is now
                                                  "PASSWORD". */

@ QUIT

  ELM                 Disconnected
  ASH                 Disconnected
  PINE                Disconnected

OK,
```

EXAMPLE OF THE PAR COMMAND

The PAR command displays current X.3 parameters, as shown in the following example. Appendix B has more information on X.3 parameters.

```
OK, NETLINK
[NETLINK  Rev. 19.3]

@ C ELM

ELM Connected
PRIMENET 19.3  ELM
@

@ PAR

FULL DUPLEX
FORWARD DATA ON:  CR  ESC  Editing  Terminators  Form  Other Cntrl
IDLE TIMER =   1.0 Secs.
ON BREAK:  Interrupt  Send indication of break  Discard output
X-OFF/X-ON ENABLED
NVT Process Control Enabled
```

```
@ SET 2:0     /*  This sets half duplex. */

@ PAR

HALF DUPLEX
FORWARD DATA ON:  CR  ESC  Editing  Terminators  Form  Other Cntrl
IDLE TIMER =   1.0 Secs.
ON BREAK:  Interrupt  Send indication of break  Discard output
X-OFF/X-ON ENABLED
NVT Process Control Enabled

@ CO

LOGIN SMITH
Password? TAXES


SMITH (user 64) logged in Wednesday, 03 Aug 83 00:49:28.
Welcome to PRIMOS version 19.3.
Last login Wednesday, 03 Aug 83 00:38:20.


Enter validation code:  @

@ PAR

HALF DUPLEX
FORWARD DATA ON:  CR  ESC  Editing  Terminators  Form  Other Cntrl
IDLE TIMER =   1.0 Secs.
ON BREAK:  Interrupt  Send indication of break  Discard output
NVT Process Control Enabled

@ SET 2:1      /* This sets Full Duplex. */

@ PAR

FULL DUPLEX
FORWARD DATA ON:  CR  ESC  Editing  Terminators  Form  Other Cntrl
IDLE TIMER =   1.0 Secs.
ON BREAK:  Interrupt  Send indication of break  Discard output
NVT Process Control Enabled

@ QUIT

  ELM                    Disconnected

OK,
```

DEBUGGING USING NETLINK

NETLINK's debugging facilities allow you to debug all kinds of X.25 circuit connection problems without the need for other diagnostic line monitors. They allow you to see exactly what X.3 parameters are sent to you from any remote host, Prime or non-Prime. You can resolve many of the common X.25 network problems with this option. For more information, see the DEBUG command in Chapter 9.

9

A NETLINK Reference

INTRODUCTION

This chapter contains information on NETLINK commands and error messages. It includes the following sections.

- A brief summary of NETLINK commands, organized by function
- An alphabetical list of NETLINK commands
- A list of error messages generated from NETLINK

To understand some NETLINK command descriptions you should have some knowledge of the X.3 and X.25 standards recommended by the CCITT, as well as some knowledge of Public Data Networks (PDNs). For special networking applications, you may need to refer to other CCITT standards and to PDN literature. Parameters to NETLINK commands are described in Appendix B. For an introduction to NETLINK, refer to Chapter 7.

COMMAND SUMMARY

This section contains brief descriptions of NETLINK commands, grouped by function. For full descriptions, see the alphabetical list in the following section, Command Reference.

Some NETLINK commands, such as the profile commands listed below, can be used as command options to either the PRIMOS-level NETLINK command or to the C, NC, or CALL commands in NETLINK.

The following basic commands, address formats, profile commands, and other commands appear on the screen when you type HELP at the NETLINK command prompt (@). (The NETLINK prompt is an at sign (@) normally, but can be changed. See Chapter 8 for an example of the PROMPT command.)

Basic Commands

Use the following basic commands to make connections to remote hosts, to switch between active connections, to return to NETLINK command mode, and to return to PRIMOS.

NETLINK Command	What It Does
C address	Connect to specified address
NC address	Connect without reverse charging
CALL address	Connect but remain in command mode
QUIT	Exit to PRIMOS
PAUSE	Exit to PRIMOS, allow 'S' to continue
CONTINUE	Continue a currently active circuit
D	Disconnect a currently active circuit

Address Formats

Address formats can be any of the following:

- system-name
- 4-10 digits
- dnic:4-10 digits
- :12-15 digit address

system-name	is a PRIMENET system name.
4-10 digits	is the numeric address of a network node.
dnic	is the Data Network Identification Code.
12-15 digit address	is the Public Data Network address.

There are two escape sequences:

(CR) @ (CR)	Escape to command mode
@ ^P	Send a ^P in the data stream

Profile Commands

These commands can be used in any of the following ways:

- As options to the NETLINK command at PRIMOS level, where they establish a default profile for all connections of the NETLINK session. For example:

```
OK, NETLINK -DNIC 2342
```

- As NETLINK commands, where they modify both the operational profile of the current circuit and the default profile for all subsequent connections. (They do not affect connections already established in this NETLINK session.) For example:

```
OK, NETLINK
[NETLINK Rev. 19.3]
@ FCTY 1 1
@ POLL 1
```

- As options to the C, NC, or CALL commands in NETLINK, where they modify only the profile of the connection about to be made. For example:

```
OK, NETLINK
[NETLINK Rev. 19.3]
@ C 405 99 -FCTY 1 1 -POLL 1
```

To be used as an option, a profile command must be preceded by a hyphen.

Operational Profile Commands: The following commands are operational profile commands. They affect NETLINK's basic operational environment.

Command	Purpose
DEbug { ON | OFF | DUMP }	Controls the debugging information display
ESCape char	Changes NETLINK escape (to a character other than @)
MODE { REMOTE_ECHO | NO_REMOTE_ECHO }	Selects remote echoing or no remote echoing
POLL n	Selects user terminal poll time interval (tenths of a second)

SPEED bps	Supplies a baud rate for hosts requiring one
TO address	Selects a destination address
TTP { id-number / name }	Selects the terminal type for certain TELENET hosts

Connect Profile Commands: The following commands are connect profile commands. They create or modify a connect profile, which NETLINK uses only while establishing a connection:

Command	Purpose
DATA text	Sets the user-data field, zero parity
DNIC dnic	Selects the 4-digit data network-id code
FCTY { CHARGE / NO_CHARGE / bytes }	Specifies up to 64 bytes for the facilities field
LDATA text	Overlays user data with zero parity in the PRID
LMDATA text	Overlays user data with marked parity in the PRID
MDATA text	Sets the user-data field marked parity
PORT n	Specifies a two-digit remote port number
PRID bytes	Specifies the 4-byte protocol-id field

Other Commands

The following are miscellaneous NETLINK commands:

Command	Purpose
STATUS	Shows active circuits
RESET	Sends an X.25 level-3 RESET
PROMPT text	Selects NETLINK's command prompt (other than @)
PROFILE [DEFAULTS n]	Displays current or default profile
SET parameter:value...	Sets X.3 terminal parameters
PAR [parameter...]	Displays X.3 terminal parameters
FILE pathname [-NTTY / -LINE / -PAD / -CPS n]	Prepares a local file transfer to a remote system
OUTFILE pathname [-NTTY / -CONTIN / -TIMEOUT n]	Copies remote host output to a local file
CLOSE { FILE / OUTFILE }	Ends a file transfer

COMMAND REFERENCE

This section describes NETLINK commands in alphabetical order.

▶ BPS bits-per-second

This command is a synonym for the SPEED command.

▶ C address [options]

The C command connects to the remote system that you specify in address. You can append an option list to this or the other connection commands, CALL and NC. The options can be any of the profile commands (listed above under Command Summary), although only the connect profile options are described here. Connect profile options and their arguments are listed below:

Option	Description
-DATA text -MDATA text	The -DATA text and -MDATA text options allow up to 12 characters of text for the User-data Field. -DATA inserts the characters with parity bits stripped while -MDATA inserts the characters with parity bits marked as received from PRIMOS.
-FCTY bytes	The -FCTY option specifies a non-default facilities field. The bytes can be up to 64 decimal numbers between 0 and 255 which are entered into the facilities field. If the option is specified without arguments, a zero-length facilities field results.
-PORT n	The -PORT option specifies a PRIMENET style port, n, between 0 and 99. It is required when the destination of the connection is a program on PRIMOS, rather than the remote login server within PRIMOS (port 0 is the default).
-PRID { bytes | text }	The -PRID option specifies either the 4 decimal bytes of the PRotocol ID field or a quoted string of 4 characters. The bytes must be between 0 and 255; the string must be enclosed in single quotes ('). Characters are inserted into the PRotocol ID field with stripped parity. This option is used when the PRotocol ID field and User-data field must be combined into a 16 byte User-data field. The data replaces the default PRotocol ID field of 001 000 000 000. This option cannot be used when the -PORT option is given because the -PORT option uses this field to pass the port number. If no arguments are given, the PRotocol ID field is reset and made available for PRIMENET.

▶ CALL address [options]

This command functions exactly as does the C command, except that after the connection has been made, NETLINK command mode continues. Input is taken from NETLINK rather than from the remote host. The CONTINUE command is required to complete the connection. The address and options are exactly as described for the C command.

▶ CLEAR { n / ALL }

This command is a synonym for the D command.

▶ CLOSE { FILE / OUTFILE }

This command ends the transfer of a file through NETLINK. It closes a local file or a remote file. Only one of each can be active at any one time.

▶ CONTINUE n

Abbreviation: CO

This command returns you to a previously connected circuit. It is used when switching between multiple connections and when returning to a connection after having given a NETLINK command. The connection number, n (value 1-6), is not needed for single connections. If n is not specified in multiple connections, the most recently active circuit is reconnected.

▶ D { n / ALL }

This command disconnects a circuit. A specific circuit (given as n) or all circuits (given as ALL) can be disconnected. If n is not specified, the most recently active circuit is disconnected.

▶ DATA text

▶ MDATA text

These commands correspond to the -DATA and -MDATA options of the C command. Whereas those options affect only a single connection, the DATA and MDATA commands establish an environment affecting any future connections.

The DATA and MDATA commands allow up to 12 characters of text for the User-data Field. DATA inserts the characters with parity bits stripped while MDATA inserts the characters with parity bits marked as received from PRIMOS.

▶ DEBUG { ON / OFF / DUMP }

Abbreviation: DE

This option displays X.3 parameter interchanges and data interchanges. DEBUG ON displays transmission or reception messages and other events for the identification of bugs or network problems. DEBUG DUMP displays additionally the contents (in octal notation) of every packet moved over the virtual circuit being debugged. The default condition is DEBUG OFF, which does not display debugging data.

▶ DNIC number

This command specifies the default Data Network Identification Code for address parsing. NETLINK supports 5 and 8 digit shorthand formats for specifying the last 8 digits of a 12-digit network address. This command controls the first 4 of the 12 digits. The text must be either null or 4 digits. A null text indicates that no DNIC is to be inserted at the beginning of the address. Otherwise the specified 4 digits are inserted as the DNIC of the address.

You can use the DNIC option in an abbreviation to set a local DNIC for non-TELENET numbers (NETLINK allows any DNIC). NETLINK's default DNIC is 3110 (TELENET).

▶ ESCAPE escape-character

Abbreviation: ESC

This command allows a user to change the escape sequence from "RETURN @ RETURN" (that is, @ at the start of a line, followed by RETURN) to "RETURN escape-character RETURN". The escape-character must be a single character. This command is useful when running NETLINK from a TELENET Packet Assembler/Disassembler (PAD) because the PAD would trap the "RETURN @ RETURN" before NETLINK has an opportunity to interpret it. For example, you can be linked to ASH through ELM and BEECH, and use the escape character to return to ELM, not BEECH.

▶ FCTY bytes

This command specifies a non-default facilities field and corresponds to the -FCTY option of the C command. Whereas that option affects only a single connection, the FCTY command establishes an environment affecting any future connections.

The bytes parameter can be up to 64 decimal numbers between 0 and 255, which are entered into the facilities field. They replace the default facilities of 001 001 002 007. The bytes are in pairs, which conforms to CCITT standards X.3 and X.29 (see Appendix B). If the option is specified without arguments, a zero-length facilities field results.

Note

In earlier versions of NETLINK, FCTY bytes were specified in octal. Now, all of NETLINK's numeric parameters are specified in decimal.

In addition, the FCTY command can have ASCII mnemonics for specific facility parameter/value byte pairs. The following mnemonics are available.

CHARGE	Sets reverse charging
NO_CHARGE	Sets no reverse charging

An "NC" command to NETLINK is the same as specifying "C address -FCTY NO_CHARGE".

▶ FILE pathname [options]

This command transfers a text file to a remote system. The text must be input to a program such as an editor or a command line on the remote host.

Input is taken from the specified file. Forwarding begins only after the CONTINUE command is issued to continue the connection with the remote system.

The following options are available:

Option	Description
-CPS n	Delays the next packet so that the average data rate does not exceed n characters per second. This option can be used with the -LINE option to overcome problems with X.25 equipment that does not properly flow control the Packet Assembler/Disassembler (PAD). Too high a data rate can be a cause of remote system character loss. The problem can be caused by remote system parameters or the X.25 equipment.
-LINE	Sends to the remote host data packets consisting of only one line of text per packet. Some non-Prime hosts accept only one line of text per packet. (Prime normally fills each packet with as many lines of text as it will hold.) Such a remote host will not properly reassemble a file transmitted with filled packets. This option forwards data a line at a time in accordance with the remote host's protocol.
-NO_SIGNAL	Inhibits the "End of file" message that usually appears in NETLINK.
-NTTY	Inhibits displaying a file that is being transferred on the terminal. Information typed at the keyboard is still printed.
-PAD	Inserts, or "pads," a space at the beginning of each null line. This option prevents an editor on the remote system from switching from INPUT to EDIT mode when a null line is encountered. (This option is necessary when using ED, and other similar editors.)

▶ HELP

This command displays NETLINK commands and a brief description of each one.

▶ LDATA text

This command is the same as the DATA command, except that the user data starts at and overlays the PRotocol IDentification field (PRID).

▶ LMDATA text

This command is the same as the MDATA command, except that the user data starts at and overlays the PRotocol IDentification field (PRID).

▶ MDATA text

This command is the marked-parity version of the DATA command. See the DATA command for a description of this command.

▶ MODE { REMOTE_ECHO | NO_REMOTE_ECHO }

REMOTE_ECHO turns on echoing and sets your terminal to half duplex. Each character that you type is echoed remotely, thus improving dramatically the performance of services such as the Office Automation System (OAS) or EMACS across a Ring.

Echoing can also be determined by an application program. With these options, NETLINK uses different packet-forwarding criteria to improve performance.

> Caution
>
> Remote echo mode can drastically increase costs over public data networks.

NO_REMOTE_ECHO turns remote echoing off and sets your terminal to full duplex. NETLINK observes normal forwarding characteristics. This is the default mode.

▶ NC address [options]

This command connects to address. It is used when collect calls (reverse charge) are not permitted, such as on international or on intranetwork connections. The address and options are the same as for the C command.

▶ OUTFILE pathname [options]

This command writes output from a remote host into a file (specified by pathname) located on the local host. This command can be used with ED to transfer files. Do an SLIST (or equivalent) command on the remote host. The following options are available:

Option	Description
-CONTIN	Appends the data to the end of the specified file without overwriting it.
-NO_SIGNAL	Inhibits the "End of outfile" message that normally appears on a timeout.
-NTTY	Inhibits the printing of output information on the terminal. Input information is still printed.
-TIMEOUT n	Causes NETLINK to close the outfile automatically if data have not been received for an elapsed n seconds. Timeout counting does not start until after NETLINK has received some data so that operator delay in giving the SLIST (or equivalent) command does not cause timeout. Because NETLINK may be talking to a non-Prime host, no protocol can be used. Therefore, data transparency cannot be provided; textual files only can be transferred.

▶ PAR [parameters]

This command displays current X.3 parameters. If no parameters are supplied, the PAR command displays text giving the meaning of the current values of the X.3 parameters (also known as the level-1 parameters).

If parameters are specified (in decimal), the PAR command displays the decimal value of those parameters. A list of X.3 parameters, called NETLINK PARAMETERS appears in Appendix B. The values of the parameters appear in the following format.

```
parameter:value parameter:value
```

Both parameters and values are printed in decimal. An unsupported parameter will have UNK or INV in the value field.

A request to read National parameters (such as TELENET's) must have the National Options Marker (NOM) preceding the parameters. Since the NOM specifying TELENET is 0:33, a PAR command to read international parameters 3 and 4 and TELENET parameter 18 would be:

```
PAR 3 4 0:33 18
```

The message:

```
Too many parameters
```

is issued if you request more data than will fit into NETLINK's internal buffer. Break up the request into two PAR commands to remedy the problem. Disabled parameters are not printed.

▶ PAUSE

Abbreviation: PA

This command returns your terminal to PRIMOS without altering the circuit state. This is useful when you want to use a PRIMOS internal command while remaining connected to a remote system. (Internal and external commands are described in the Primos Commands Reference Guide.) Use the START command to reenter NETLINK command mode after a pause and NETLINK's CONTINUE command to resume the connection.

Note

Be sure not to issue any external PRIMOS command, or your NETLINK session (and your remote connection) will be destroyed. There is a list of commands in the PRIMOS Commands Reference Guide that identifies each command as external or internal.

▶ POLL n

This command sets the polling time interval. The value n specifies the tenths of a second between polls of the terminal. NETLINK checks your input buffer for new characters every 0.n seconds. The default value for n is 5, or a polling rate of twice a second. Frequent polling (a low value of n) speeds up terminal response, but requires more packets. Infrequent polling (a high value of n) reduces the number of packets transmitted; this lowers the cost of the connection, but makes terminal response more sluggish.

▶ PORT n

This command specifies a PRIMENET port, setting the profile so that subsequent connections are addressed to an application program waiting on the specified port, rather than to the remote login server, which uses port 0 as the default. The value of n must be from 0 to 99.

▶ PRID { n / text }

This command establishes the PRotocol IDentification (PRID) field data for future connections. It specifies either the 4 decimal bytes of the protocol identification field, or a single-quoted string of 4 ASCII characters. The bytes must be between 0 and 255; the text string must be enclosed in single quotes ('). Text characters are inserted into the PRID field with stripped parity.

This command can be used when the PRID field and User-data field must be combined into a 16 byte User-data field. The data replaces the default PRID field of 001 000 000 000.

This command cannot be used when the -PORT option or command is given because -PORT uses the PRID to pass the port number. If no arguments are given, the PRID field is reset and is made available to PRIMENET.

Note

In earlier versions of NETLINK, PRID bytes were specified in octal. All of NETLINK's numeric parameters are now decimal.

▶ PROFILE

Abbreviation: PRO

This command displays the profile of the current circuit. If you are not connected to another system, the message "Not Connected" appears.

▶ PROFILE DEFAULTS

Abbreviation: PRO DEF

This command displays the default profile to be used for all new connections.

▶ PROMPT text

This command changes the NETLINK command mode prompt. The text can be up to 32 characters in length, and is displayed with a trailing space. The text replaces the current prompt. This is useful when you have logged into a remote host that displays the same prompt as NETLINK (@). Changing NETLINK's prompt reduces the confusion when you exit from NETLINK command mode.

▶ QUIT

Abbreviation: Q

This command returns you to PRIMOS from NETLINK. Any active connections are disconnected.

▶ SET parameter:value parameter:value

This command sets the X.3 parameters (also known as the level-1 parameters) that control terminal characteristics. Parameters and their values are specified in decimal. NETLINK supports X.3 parameters for international and PDN connections. A complete description of X.3 parameters can be found in documentation supplied by your PDN administration (such as TELENET or DATAPAC).

▶ SPEED bits-per-second

This command (for which BPS is a synonym) tells NETLINK how to respond to terminal speed requests from the remote host. Some hosts hang unless you provide a valid terminal speed. Most hosts do not expect a value greater than 1200. This command sets up X.3 parameter 11.

▶ STATUS

Abbreviation: STAT

This command displays the status of all active circuits or the message "No Active Circuits." The displayed information includes NETLINK circuit number, address, time (in hours and minutes) that the circuit has been open, and traffic information (number of packets sent and received). The arrow indicates the currently active circuit in multiple connections. For example:

```
@ STATUS

  Circuit #        Address          Time     Pkt in    Pkt out
  ---------  ----------------       ----     ------    -------
       1     BIRCH                  6:11        425        233
  ->   2     ASH                    3:32         13          6
       3     LINDEN                 2:01        337        158
       4     311061700702           1:20         18          6
```

▶ SW n

This command is used in multi-connection operations. It allows you to switch the current circuit to circuit n. Your terminal remains in command mode and you must issue a CONTINUE command to connect to the specified circuit. The command "CONTINUE n" is identical in results to "SW n" followed by "CONTINUE".

▶ -TO address

This command establishes a destination address. As an example, "C address" and "C -TO address" are the same command. If either is entered as a command to NETLINK, then a subsequent C command, without an address, connects to the default address.

TO is most useful as -TO on the PRIMOS command line invoking NETLINK, because it avoids command mode entirely. This allows construction of abbreviations (using the PRIMOS ABBREV command), such as:

```
OK, ABBREV -ADD_COMMAND DENVER NETLINK -TO DENVER
```

When -TO is used this way, NETLINK will perform an automatic "QUIT" if the connection is broken. This allows you to connect to remote systems without ever entering NETLINK's command mode. For example:

```
OK, DENVER
[NETLINK Rev. 19.3]

DENVER Connected

                    /* user session with remote system
                    /* user logs out

DENVER Disconnected

OK,                 /* you are now on your local system
```

Additional options can be used with -TO as required. The following example is the equivalent of using the NC command and creates an abbreviation for an otherwise lengthy command:

```
OK, ABBREV -ADD_COMMAND FARGO NETLINK -TO 701884 -FCTY NO_CHARGE
```

▶ TTP { id-number | name }

This command tells NETLINK the terminal type. This command affects only TELENET and a few types of hosts. Those hosts "read the terminal type" and perform their own echoing and carriage control. Allowed terminal names are: UNKNOWN, BEEHIVE, FOX, OWL, PRINT, PT45, and VT50.

Names are translated into the appropriate value for TELENET parameter 23. If a number is specified, then this is used as the value for TELENET parameter 23.

ERROR MESSAGES

Any of several error conditions can occur while you are making or using a NETLINK connection. For example, the remote host or the data connection can fail. Error messages displayed by PDNs and by PRIMENET through NETLINK are listed below.

Error Messages Generated From PDNs

These error messages may appear when you try to connect to a system on a PDN.

- address Access Barred

Access is not allowed from your address.

- address Busy

All available circuits to the remote host named address are busy. Try again later.

- address Invalid Call

There is an error in the call request packet, or the host PDN does not accept calls of this type.

- address Local Procedure Error

A network protocol error has been encountered. Try again later. If the problem persists, inform your computer center staff.

- address Network Congestion

Temporary network problems exist. Try again later. If the problem persists, inform the computer center staff. This message indicates problems with the local network or with a Public Data Network.

- address Not Obtainable

There is no path available to the remote host. It may indicate an incorrect host address.

- address Out of Order

The remote host is currently not available to network users.

- address Refusing Collect Call

You cannot make collect calls to address. Use the NC command.

- address Remote Procedure Error

A network protocol error has been encountered. Try again later. If the problem persists, inform your computer center staff.

PRIMENET Error Messages

These messages may appear when you are connecting to a PRIMENET node, or during the life of a connection.

- address Disconnected number (Network Server logged out)

The networks have been shut down on your local system with the STOP_NET command. NETLINK logs you off of the remote system that you are on.

- Host down

The line to the remote host is down.

- Illegal address

The address that you specified is not a legal network address. Type "STATUS NETWORK" at PRIMOS level to see a list of correct network names and/or addresses.

- Network Server logged out

NETMAN, the network server, stopped. It logs out, shutting off network services and loopback services. This message also appears if the operator stops NETMAN.

- No remote users

Remote users are not allowed to log in to the system that you specified in your address. Usually this means that the maximum number have already logged in and there is no room for another user.

- Port not assigned

The port that you specified has no process waiting to receive calls.

- Rejecting

For PRIMENET nodes, this means that the remote node is not in your system's address tables. See your System Administrator to find out if that node has been configured.

For non-PRIMENET hosts or user ports (port-ids between 1 and 99), this indicates that the receiving process or machine rejected the user's request. For example, a non-Prime host might look at your remote login call request and reject it because of unacceptable facilities.

- System busy

Usually, this message indicates a lack of resources that are required for the establishment of the virtual circuit. Try again later.

- System not up

The remote system is down.

- Timeout on call request

Your call request to the remote node was timed out. This indicates that the other node or the network (especially if it is a PDN) is congested or down. Try again later.

- Timeout on clear request

Your call request to the remote node was timed out when you tried to clear the circuit. Generally, this isn't something you should worry about. You may leave a virtual circuit established, but when you log out your end of the virtual circuit is guaranteed to disappear.

- Timeout on reset request

This message appears if PRIMENET tried to resend a series of data packets that were sent out of sequence. The Remote File Access manager automatically retries data transmission. This results in the "CIRCUIT RESET" message in NETLINK. Further data transmission may be possible. Data may have been lost, unless you placed some higher level protocol around it. If not, you may have to clear the circuit and try again.

Route-through Error Messages

The following section describes each message that may occur from Route-through errors.

- Routed-Thru call request looping

A Route-through call request is looping because of a mismatch in network configurations over a network. Route-through operates by establishing a series of virtual circuits between intermediate nodes along its path, and partially because these paths are statically determined at PRIMENET configuration, it is possible that a path between two nodes could contain a loop. This loop can be detected, but not avoided or removed. Ask your System Administrator to resolve the problem.

- Routed-Thru circuit timeout

This indicates that an acknowledgment to a call request (circuit establishment) over a Route-through path (involving one or more gateway nodes) has not been received in a short enough period of time. This indicates that one or more of the intermediate gateway nodes or networks is badly congested or has suffered a crash. It can also indicate that the Route-through path is extremely long. Try again. If this persists, call your System Administrator.

- Route-Thru: Not enough memory

The buffer area allocated by the Route-through server to perform the exchange of messages over the pair of virtual circuits that make up each Route-through virtual circuit was exhausted, or the Route-through server was congested. Try later.

- Route-Thru protocol error

Some intermediate gateway is down, or has violated the intergateway protocol during the establishment of the end-to-end virtual circuit, which is cleared. Try again.

- Route-Thru server down

In the circuit establishment or traffic phases, this indicates that every path to the remote node is down. Specifically, at some intermediate node, the Route-through server (distinct from the Network Server) has logged out. This can also happen in the middle of the virtual circuit traffic. In either case, the Route-through server may have been logged out by the operator. Try again later, or contact the local or remote System Administrators to resolve the problem.

PART V

Advanced PRIMENET

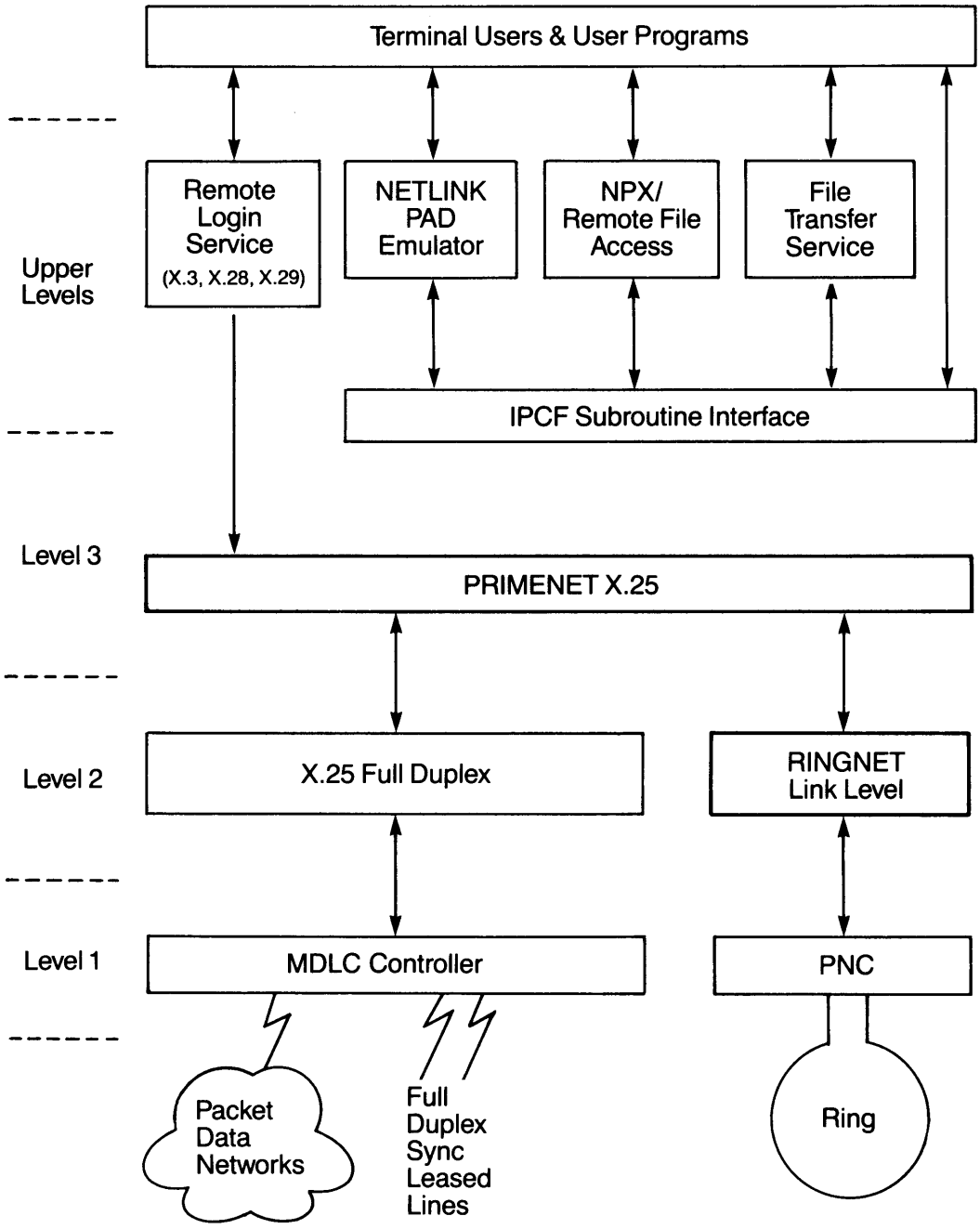
10
PRIMENET Architecture

INTRODUCTION

PRIMENET is made up of several layers, as shown in Figure 10-1. This layered structure is based on the International Standards Organization Open Systems Interconnection (ISO OSI) model, to make the 50 Series systems able to communicate with all other systems that support X.25 protocols. Each layer is described in the next section.

PRIMENET'S LAYERS

Each of the functional layers of PRIMENET has its individual functions and each interfaces with the adjacent layer or layers.



Layers of PRIMENET Architecture
Figure 10-1

Level 3, the packet interface, creates and controls virtual circuits across the network, handles error recovery, and controls the flow of information. It also keeps track of the process to which each packet is being transferred. You can write network-based programs using the Interprocess Communications Facility (IPCF subroutines), which are described in Chapters 13 through 16. These routines interface directly to level 3. Level 3's X.25 support provides a standard interface to upper-level software no matter what kinds of links make up Levels 1 and 2.

Level 2, the link protocol level, corresponds to the ISO OSI's data link layer. It describes a protocol to which two linked nodes must adhere when they transfer information from one to the other. This protocol dictates the format of the data; how the nodes should request, transfer, receive, and acknowledge the data; and how to signal faulty transmissions, should any occur.

Level 1 is the hardware interface. It is the equivalent of the ISO OSI's physical layer. This layer acts as an intermediary between the physical transmission medium (twin-axial cable or transmission line) and the rest of PRIMENET and the system. Depending on the type of network, one of three controllers governs action at this level. These controllers are the PRIMENET Node Controller (PNC), the Multiple Data Link Controller (MDLC), and the Intelligent Communications Subsystem Model 1 (ICS1), described later in this chapter. These controllers are intelligent (have memory) and implement some of the level 1 and 2 protocols.

PRIMENET's upper layers provide user services. These internal functions operate on the lower layers to perform actions directly specified by you or actions that facilitate completion of user tasks. These services use the same IPCF subroutines that are mentioned above.

ADVANTAGES OF LAYERED ARCHITECTURE

The layered approach described above has several benefits. The simplicity and structured design is the same no matter how many systems are linked in a network. In addition, since PRIMENET supports internationally recognized standards such as X.25, X.3, X.28, and X.29, you can easily link to any other network, Prime or PDN, that supports the same standards. Thus, Prime systems can be integrated easily with existing equipment.

You interact with only the top layers of PRIMENET. Since these top layers interact with the lower levels, you do not have to know anything about the physical connections between linked systems, or how data are formatted and checked during transfer.

In many cases, you need not even know that network connections are being used. For example, remote files can be accessed in exactly the same way as local files (see Chapters 2 and 3 in this book).

Another advantage of the layered architecture of PRIMENET is that any changes made to the lower levels are transparent to you. Thus, enhancements of the lower levels need not change how you invoke or use PRIMENET.

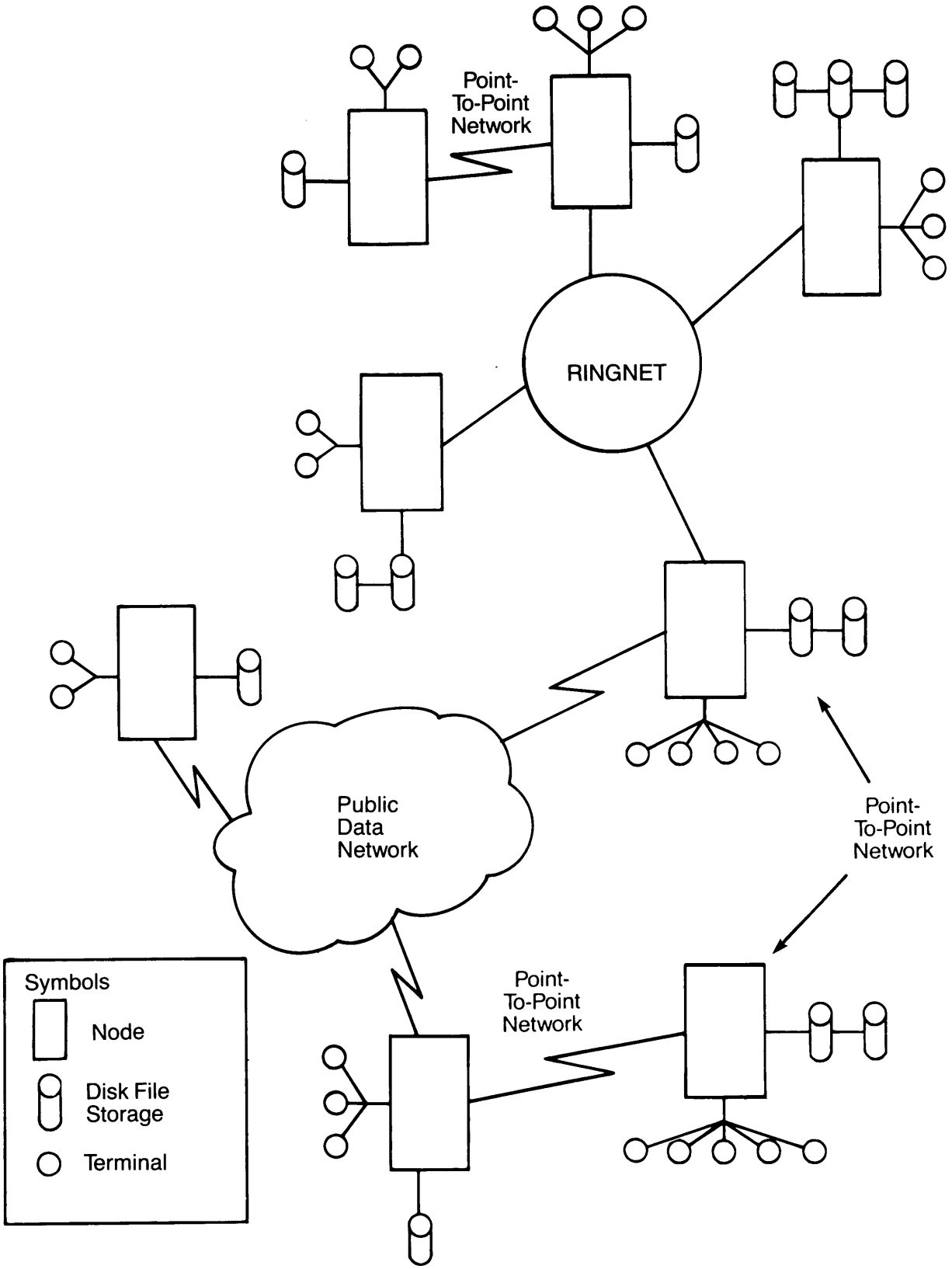
NETWORK TYPES

PRIMENET supports the following types of network links:

- Local Area Networks in a Ring (RINGNET)
- Point-to-point Connections
- Public Data Network (PDN) Connections
- Route-through Connections

These links allow you to log in to remote systems or access remote files from their local systems. Figure 10-2 shows typical examples of the three types of network links.

For the rest of this chapter, the word node represents any system that is linked to others in a network.



Examples of Networks
Figure 10-2

Prime's Local Area Network: RINGNET

RINGNET is Prime's Local Area Network (LAN). Many nodes can be connected through a high-speed, one-way, serial-synchronous twin-axial cable. (See Figure 10-3.)

Each node in a ring network is equipped with:

- A junction box, which connects the twin-axial cable to the computer. The junction box has a passive relay that switches to "pass-through" if it detects a power failure on the PNC. This preserves a ring's integrity.

- A PRIMENET Node Controller (PNC) board, which controls the ring protocol and the flow of data between ring nodes.

RINGNET uses a token ring protocol. A special bit pattern called a token circulates continuously around the ring. A node cannot transmit data until it detects the token. When it can transmit, a node issues a packet containing a 4-byte header and from 4 to 2044 bytes of data through its PNC.

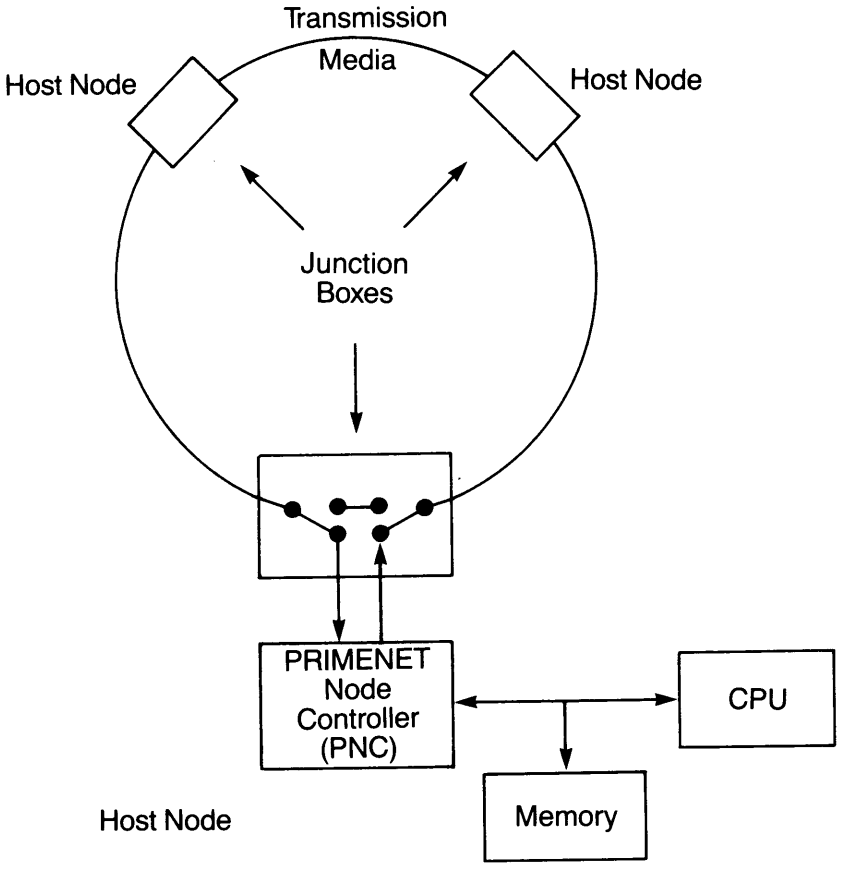
The packet circulates around the ring, at a data rate of eight megabits per second, to the destination node. There, the destination PNC receives the packet and sets a flag in the packet to acknowledge the receipt. The packet travels around the rest of the ring back to the source PNC, which does the following:

- Removes the packet from the ring

- Passes on the token

- Checks the acknowledgement flag

- Interrupts the source operating system to signal successful transmission



RINGNET Configuration
Figure 10-3

Each PNC acts as an active data repeater for packets between other ring nodes (see Figure 10-3). Node B's PNC handles data transmitted between Node A and Node C. This repetition requires no software intervention on the part of Node B; its PNC firmware handles the data transmission.

The PNC uses Direct Memory Access (DMA) I/O to transfer up to two Kbytes per packet. It needs only one interrupt per data packet to indicate a node's success or failure in receiving or transmitting a packet.

RINGNET serves all nodes in the ring equally, so that one system cannot monopolize the network. It automatically checks all packets for integrity, and does not require user intervention or separate acknowledgement messages. In addition, RINGNET offers you the ability to expand into larger networks without suffering any degradation of performance.

If any of the nodes in a ring network are powered down or broken, the rest of the nodes are still able to send data around the ring. The junction box in a disconnected node allows messages to pass through without interruption to the next node in the ring, as long as the cable distance between adjacent active nodes does not exceed 750 feet (230 meters).

Point-to-point Connections

Nodes in a point-to-point network communicate through dial-up or leased telephone transmission lines. Each node can contain one or two MDLCs; each MDLC can support two or four lines at standard modem speeds.

Alternatively, each node can support an Intelligent Communications Subsystem ICS1 controller. The ICS1 acts effectively as a front-end computer that supports both synchronous and asynchronous serial communication lines. It has an onboard microprocessor, 65Kb of downline-loaded RAM, PROM-resident diagnostics, and its own operating system.

Information travels from the processor of one node, through the MDLC or ICS1 and a modem, across the transmission lines to the modem and MDLC or ICS1 of another node.

PRIMENET uses the X.25 High-level Data Link Control (HDLC) protocol on all synchronous lines and controllers. For leased-line communications, the MDLC supports bisynchronous (BSC) or HDLC framing. The Intelligent Communications Subsystem, Model 1 (ICS1) supports HDLC framing only.

Note

> At PRIMOS revisions previous to Rev. 19.3, the MDLC also supports HDX, a modified HDLC protocol, to allow half duplex communications across dial-up lines. For more information on half duplex communications, refer to the Rev. 19.0 System Administrator's Guide.

It is recommended that all synchronous PRIMENET lines (either point-to-point or to a Public Data Network) use High-level Data Link Controller (HDLC) framing and Link Access Protocol Balanced (LAPB) protocol, if possible.

The Multiple Data Link Controller (MDLC) and the Intelligent Communications Subsystem Model 1 (ICS1) can be used simultaneously by several processes (PRIMENET-related or other system processes).

Like the PRIMENET Node Controller (PNC), the MDLC uses DMA I/O to transfer data to and from main memory, and issues an interrupt to the operating system only after an entire packet of data has been received or transmitted. It handles all error checking and frame formation in firmware.

The ICS1 acts as a front-end processor that allows the main processor to off-load communications functions. This provides greater performance and lower overhead.

Public Data Network (PDN) Connections

Prime systems can subscribe to all Public Data Networks (PDNs) that support the CCITT X.25 protocol standard. Supported PDNs include TELENET and TYMNET in the United States, DATAPAC in Canada, IPSS in Great Britain, TRANSPAC in France, and DATEX-P in Germany.

All of these networks transfer and process information in packets. PDNs often provide service at substantial savings over methods requiring dedicated transmission lines or dial-up circuits.

With PRIMOS Revision 19.3, you can subscribe to more than one Public Data Network. This is a new feature with 19.3. For more information on PDNs, refer to Chapter 12.

Route-Through Connections

Route-through is a facility that permits Prime-to-Prime nodes, not directly connected, to communicate with each other through gateway nodes. Route-through is transparent to the user. This process is handled by the Route-through server, which handles the call requests, allocates the necessary virtual circuits, and routes the message to the destination node.

11

Ports and Virtual Circuits

INTRODUCTION

If you want to write network programs that use Interprocess Communication Facility (IPCF) subroutines, you need to know about

- Ports
- Virtual circuits

In most cases, you don't need to be concerned with ports when you use PRIMENET facilities such as Remote Login and NETLINK. Ports and virtual circuits are handled automatically and therefore are transparent to you. This chapter describes the concepts of ports and virtual circuits with which you'll need to be familiar if you do want to write network programs.

Some of the information in this chapter regarding virtual circuits (in particular, the sections Virtual Circuit Status Array and Clearing Codes) assumes a knowledge of Inter-Process Communications Facility (IPCF) subroutines and their arguments. These are described in Chapter 14. Refer to Chapter 15 for a step-by-step description of how to establish a virtual circuit. Chapter 16 describes IPCF programming strategy.

PORTS

A user process or program that uses IPCF subroutines to communicate with a remote system must have some way of identifying both the remote node and the destination process on the remote node. A standard X.25 node address can identify the remote node, but it cannot identify to which of the 255 possible destination user processes to attach. Processes are identified by means of ports within a node, which are specified by using network subaddresses.

Ports, Programs, and User Processes

Each node in a PRIMENET network has a pool of available ports that may be assigned by programs running under PRIMOS (see Figure 11-1). Each port represents a target to which incoming network connection requests may be routed.

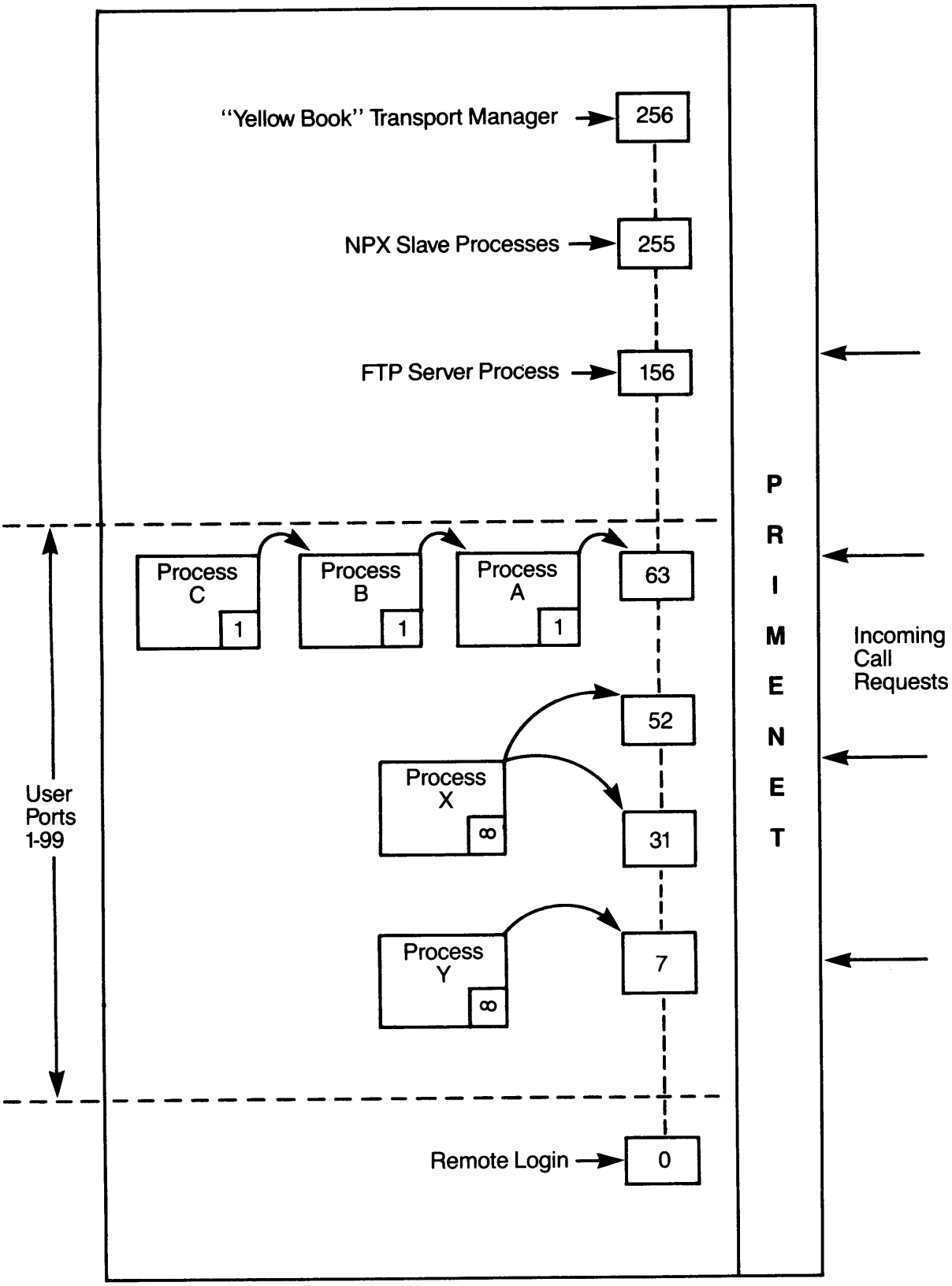
The assignment of a particular port (see X$ASGN in Chapter 14) declares a program's interest in receiving the connection requests addressed to that port. Once a program has been notified that such a request is pending on a port it has assigned, it may use other network subroutines to get information about the originator of that request and to complete establishment of the connection.

A process may assign a port permanently, or it may assign it with the provision that it be automatically unassigned after a certain number of connection requests have been handled. A process assigning an already-assigned port is placed on a queue for that port, behind the processes that have already assigned the port. An assigning process passes a numeric parameter to indicate whether it wants to handle all calls to that port or a specific number of calls. For example, PROCESS Y in Figure 11-1 will be able to handle eight calls, while PROCESS C will be able to handle only one call. A negative value tells the port to keep that process at the back of the queue.

Port Assignments

The ports available to PRIMENET user processes are 1 to 99 inclusive. Port 0 is permanently reserved for PRIMENET's remote login. Ports 100 to 255 are reserved for PRIMENET's internal use. For example, port 253 is permanently reserved for the Route-through Server. (You should also note that FTS servers have configurable ports, which occur in the 1-99 range.)

Ports are a system-wide resource that must be administered on a system-wide basis, to avoid undesired interference between user applications.



The Port Mechanism
Figure 11-1

Using the PRIMENET IPCF subroutine named X$ASGN, you can assign a port. You must specify a port number for the call to go through to the correct destination process. If you do not specify a port number, the call is sent to the default port, which is the remote login port (Port 0).

When several processes wait in a port queue, the automatic unassignment of one process allows the next process in the queue to handle subsequent requests.

To manage the bottom of the queue, and to ensure that every request gets handled, it is possible for a process to permanently assign itself at the end of the queue. The process remains watching the port forever (until it unassigns itself from the port, or logs out), and handles requests only when other processes are unavailable.

Note

A process that only makes calls, and does not receive them, need not assign a port.

VIRTUAL CIRCUITS

When one process specifies the node and port of another process, PRIMENET establishes a bidirectional link between the two through a virtual circuit (VC). A virtual circuit is a logical path from one process to another across the network.

Virtual circuits do not necessarily correspond exactly to physical communication lines. For example, one physical line may carry many virtual circuits, just as one telephone trunk line may carry many voices at the same time. Moreover, a virtual circuit between two given systems might use a different physical path each time it is established (if different physical paths exist).

Each virtual circuit has an identifying number to distinguish it from all others. PRIMENET currently allows up to 63 virtual circuits at a time on each node. One VC across a PDN may traverse many physical links and nodes.

Intra-node Calls

In addition to being able to create virtual circuits to ports on remote nodes, you can create a virtual circuit to ports on the local node. You do this by specifying the local node as the destination node in the call connect subroutine. A virtual circuit created this way behaves exactly the same as one that is created to a remote node.

This feature is useful for developing and testing network applications because it permits the testing of several different pieces of a distributed application on the same node. It is also useful because it permits local users to access a system the same way that remote users do.

Passing Off Virtual Circuits to Other User Processes

PRIMENET permits an application to switch a virtual circuit over to another application. Detailed information is found in the description of X$GVVC/XLGVVC in Chapter 14.

Virtual Circuit Status Array

Each process holding virtual circuit connections must specify a two-word (two full words) array for each virtual circuit's status. When needed, PRIMENET reports status changes for the virtual circuit into that array.

The caller specifies this virtual circuit status array in the call to the connect routine (X$CONN, X$FCON, XLCONN). The call receiver specifies the array in a call to the accept routine (X$ACPT, X$FACP, XLACPT). The virtual circuit status array may be written into by PRIMENET until the virtual circuit is cleared. (The clear confirmation is written into the array as well.)

The first of two array words is continually updated by PRIMENET to reflect the latest status of the circuit. When a data transmit or data receive completes, this virtual circuit status word is updated to the value XS$CMP. In addition, the status argument of the X$TRAN/X$RCV call that initiated the now-completed action is also set to XS$CMP.

Should the circuit ever be reset because of errors in the communications media or through network congestion, the virtual circuit status word is updated to the value XS$RST. In addition, the status arguments of any pending data transmits or data receives are also set to the same value.

When a user-held network connection is disconnected (cleared), the first word of the virtual circuit status array is set to the "circuit cleared" status code (XS$CLR). At this time, the second word of the array is also valid, and indicates the reason for the clearing. (See the following section, Clearing Codes.)

The purpose of the virtual circuit status array is to provide processes an easy way to poll the state of their (multiple) virtual circuits. A process that is managing many virtual circuits, each with several data-moving operations in progress, need only poll the virtual circuit status arrays for the completed status (XS$CMP) to see which circuit(s) are ready for more traffic. The user process should change the virtual circuit status to XS$IDL or XS$IP from XS$CMP to wait for more data transfers.

Note

PRIMENET data communication is handled by a special user process, NETMAN, that is classified as user type 'NSP'. This process runs at high priority, and thus its activity might interrupt a program during its execution. An interruption affects the program's handling of the virtual circuit status array and other returned status variables. You should never do a repeated test against any of these returned statuses, but first obtain your local copy, to ensure it remains unchanged during the test.

Clearing Codes

A virtual circuit may be cleared by a Public Data Network (PDN), by PRIMENET, or by either of the two processes controlling it. The reason for clearing appears in the second word of the virtual circuit's status array. The high-order byte of this word is the clearing cause and the low-order byte is the diagnostic code byte.

Clearing causes can be network-generated, or requested by you ("DTE originated" in X.25 terminology), and assigned a specific clearing cause. There are several network-generated causes defined in the X.25 standard. They are sometimes referred to as call progress signals, since they give information about "how far a call has progressed when cleared."

The clearing cause is most likely one of the CC$xxx codes listed below, but it can also be another cause defined by a national network. If the clearing cause is CC$CLR, the circuit was cleared by either PRIMENET or a user process ("DTE-originated"). In any case, the diagnostic code may contain useful information. The X.25 standard defines several diagnostic codes in its Appendix 5. Individual public data networks may have defined further codes. You can use clearing codes with the X$KEYS insert file that PRIMENET provides.

When a call is cleared explicitly with XCLR/XFCLR, the clearing user process may specify the value of the diagnostic code in the why argument (see X$CLR/XLCLR in Chapter 14). Communicating processes may use this facility to describe fatal error conditions or supply final status information. Although the entire range 0 to 255 may be sent, you should limit codes to the range 0 to 127. When PRIMENET clears a circuit, the diagnostic code is one of the CD$xxx codes listed below. The values 128 to 255 are reserved for future use by PRIMENET.

The list of status codes that may be written into the first word of the virtual circuit status array appears in the sections on X$CONN/X$FCON/XLCONN and X$ACPT/X$FACP/XLACPT in Chapter 14. The following table lists predefined clearing causes (CC$xxx) and diagnostic codes (CD$xxx) for the second word of the array:

Clearing Cause	Meaning
CC$CLR	This circuit was explicitly cleared. There might be a diagnostic code from you or PRIMENET.
CC$BAD	The call request packet is invalid.
CC$BAR	Access to the requested system has been barred.
CC$BSY	The called system is not accepting connections right now.
CC$DWN	The system to which this circuit is connected is not currently operating.
CC$LPE	Local procedure error. (See CC$RPE.)
CC$NET	Temporary packet network congestion.
CC$NOB	The requested system is not obtainable through the packet network.
CC$RPE	Remote procedure error. Violation of X.25 protocol through a packet network.
CC$RRC	The requested system refuses a collect call.

Diagnostic Code	Meaning
CD$BSY	The target system cannot accept any more connections at this time.
CD$DWN	The system to which this circuit is directed is not currently operating.
CD$IAD	A connection request specified an unknown or illegal address.

CD$LOP	A Route-through call request is looping.
CD$LPE	A violation of the X.25 protocol has been detected.
CD$MEM	The Route-through server does not have enough memory for a call to be routed.
CD$NRU	The target system has no more remote processes available at this time. (Used with remote login.)
CD$NSV	The PRIMENET server process is not running.
CD$PNA	The port to which this call is directed is not assigned in the target system.
CD$RTD	The Route-through server is down or inconsistencies exist between different network configuration files.
CD$RTE	A Route-through protocol error was detected.
CD$SNU	The target system is not yet available for login. (Used with remote login.)
CD$TCA	A call request has not been answered by the target node, so the circuit was cleared.
CD$TCR	A clear request has not been duly confirmed by the target node, so the circuit has been cleared and dropped.
CD$TMO	The Route-through server experienced a virtual circuit timeout.
CD$TRS	A reset request has not been duly confirmed by the target node, so the circuit has been cleared.

The clearing causes shown match the masked (not shifted) high-order byte of the second word of the virtual circuit status. A FORTRAN test would be:

```
IF (AND(VCSTAT(2),:177400) .EQ.  CC$CLR) ...
```

Other essential features relating to IPCF subroutines are described in Chapter 14.

12
PRIMENET and PDNs

INTRODUCTION

Prime 50 Series systems can connect to all Public Data Networks (PDNs) that support the CCITT X.25 protocol standard. Supported PDNs include TELENET and TYMNET in the United States, DATAPAC in Canada, validated for PSS in Great Britain, TRANSPAC in France, and many others. PDNs transfer and process information in packets. Packet-switching networks make it possible for several users to share the same equipment simultaneously. These networks can often provide service at substantial savings over methods requiring dedicated transmission lines or dialup circuits.

The following sections describe how PRIMENET allows you to conveniently and easily communicate to PDNs.

PDNs and NETLINK

Prime systems can talk to non-Prime systems on PDNs with NETLINK, which is described in Chapters 7, 8, and 9. All you need to know is the address and the login sequence codes of the host in order to access it with NETLINK over the PDN. NETLINK allows you to communicate over any X.25 network to which the local system is linked. NETLINK emulates a PDN packet assembler/disassembler (PAD). It converts the asynchronous terminal output into X.25-formatted packets of information that can be transmitted over the network.

If your system has a PDN link, NETLINK allows access to any system in the network, both Prime and non-Prime. You don't have to log out of the local system to invoke NETLINK; in fact, NETLINK supports simultaneous links with up to six remote systems and lets you move between them and your local system at will. This capability puts the wide variety of PDN facilities within quick reach of any Prime user in a network.

EASE OF ACCESS

Because PRIMENET uses the X.25 protocol, a connection across a PDN functions similarly to a normal PRIMENET connection. No special protocol conversion or other processing is required.

The user of a Prime 50 Series system linked in a PDN has access to all other members of the PDN. This means that any Prime terminal user can access all other member systems, both Prime and non-Prime, and that all PDN terminal users can access the 50 Series system. It is very important to implement security controls in this environment. The Network Planning and Administration Guide describes how a Network Administrator can configure the proper security controls for a network.

MULTIPLE PDN SUPPORT

A Prime node can support multiple PDNs. For example, PRIMENET can connect to a different PDN on each synchronous line; your system could connect to both TELENET and TYMNET. However, PRIMENET cannot support more than one line to the same PDN.

ROUTE-THROUGH

Route-through is a facility that permits Prime-to-Prime nodes, not directly connected, to communicate with each other through gateway nodes. Route-through is transparent to the user. This process is implemented by the Route-through server, which handles the call requests, allocates the necessary virtual circuits, and routes the message to the destination node.

A gateway node can serve as a link between two areas in a network, for example, between RINGNET and a PDN. Thus, users on any system in that RINGNET could communicate with the PDN directly. Because the Network Administrator can configure alternate paths to a system through the Route-through facility, data can be sent between networks that are connected through a gateway system across a PDN.

PART VI

PRIMENET Programming

13

Introduction to Network Programming

INTRODUCTION

PART VI of the PRIMENET Guide deals with network programming. The following subroutines are described.

- Inter-Process Communications Facility (IPCF) subroutines
- The File Transfer Service (FTS) FT$SUB subroutine

Refer to Appendix A for X.25 programming guidelines. The following sections briefly describe the chapters contained in this part.

IPCF SUBROUTINES

Chapter 14 contains an overview of the IPCF subroutines, an explanation of the IPCF naming conventions, a functional summary of the subroutines, and a detailed description of the subroutines and their arguments.

IPCF Programming Examples

Chapter 15 provides examples of applications that use IPCF subroutines. The examples show

- How to establish a virtual circuit link
- The general layout of an IPCF application
- A sample network file-transmission application
- An example of the use of Fast Select in an application

IPCF Programming Strategy

Chapter 16 describes the basic principles of network programming, using IPCF. The following concepts are described.

- Front-end processes
- Servers
- Performance
- Window and packet sizes in virtual circuits
- Return codes
- Network event waiting
- Ending a network program
- The effects of network shutdown and startup on applications that use IPCF subroutines

FTS PROGRAMMING WITH THE FT$SUB SUBROUTINE

Chapter 17 describes the FT$SUB subroutine, which provides a programming interface to the FTS utility FTR. Network users can use FTR to transfer files between machines that are connected in a network. Programmers can use FT$SUB to create programs that perform repetitious procedures, such as daily file transfers. FT$SUB performs the same group of tasks as the FTR utility, which includes submitting, monitoring, and canceling user file transfer requests.

14

IPCF Subroutines

INTRODUCTION

This chapter describes the set of subroutines that make up the PRIMENET Interprocess Communications Facility (IPCF).

This chapter has four parts:

- IPCF Overview
- Naming conventions
- Summary of subroutines
- Detailed descriptions of subroutines

To use IPCF subroutines, you need a knowledge of ports, virtual circuits, and a basic understanding of IPCF concepts. Chapter 11 describes ports and virtual circuits, and clearing codes. Chapter 15 shows you how to establish a virtual circuit, describes the general layout of an IPCF program, and contains two longer programming examples. Chapter 16 discusses programming strategy.

IPCF OVERVIEW

IPCF subroutines are designed for use by programs running in V-mode or I-mode on Prime 50 Series computers. These subroutines can be called from any high level language. With IPCF subroutines, you can develop modular applications. Each module can run as a separate application from others on the same PRIMENET system. The different modules use the IPCF subroutines to establish a connection between each other. This connection is called a virtual circuit.

IPCF subroutines enable applications to send or receive messages to systems within a PRIMENET network or between processes on the same system (intra-node communication is discussed in Chapter 10). Two forms of each IPCF subroutine are available: a simple version with a short parameter list (short form) and a full version with a larger number of parameters (long form). The short forms can be used without a detailed knowledge of X.25 protocol. Because most of the X.25 protocol is automatically handled by PRIMENET, you can use the short forms for Prime-to-Prime connections. All the information you need in order to use the short forms is contained in this manual.

The long forms complement the short forms. With the long forms, you can use X.25 functionality fully. The long forms are most useful in handling Prime-to-non-Prime connections. In addition to understanding the basic concepts of IPCF subroutines, you must know the 1980 X.25 protocol to use the long forms. Appendix A contains information about X.25.

NAMING CONVENTIONS

Generally, the names of IPCF subroutines begin with three types of prefixes: X$, X$F, and XL. The X$ prefix indicates one of two things. It prefixes a subroutine that has only one form or is the short form of a particular subroutine. The short form subroutines use a minimum argument list. For example, there is only one wait subroutine, X$WAIT. While, on the other hand, X$CONN is the short form of the call request subroutine.

The X$F prefix indicates short-form subroutines tailored for the fast select facility as defined in the X.25 standard. X$FCON is the fast select version of the connect subroutine. Finally, the XL prefix indicates a long form subroutine. For example, XLCONN is the long form of the call request subroutine.

SUMMARY OF IPCF SUBROUTINES

The PRIMENET subroutines described below can be called by any application that runs as a V- or I-mode program. The library VNETLB contains references to these subroutines. The file SYSCOM>X$KEYS.INS.FTN contains the definitions of key and error codes for FORTRAN. The file SYSCOM>X$KEYS.INS.PL1 contains those for PL/1 Subset G.

Note

The integer default in FTN is INTEGER*2; in F77, it is INTEGER*4. To avoid confusion and incorrect results, type all integers explicitly.

The SYSCOM>X$KEYS insert file is kept for compatibility with old program source files. It is a copy of the SYSCOM>X$KEYS.INS.FTN file.

The subroutines are listed below by function; the short forms for each functional group are noted.

Subroutine Name	Function
X$ASGN	Declares an application's interest to receive a specific class of incoming calls (assigns a port).
X$CONN X$FCON XLCONN	Requests a virtual circuit connection.
X$GCON X$FGCN XLGCON	Provides information about incoming calls.
X$ACPT X$FACP XLACPT	Accepts a connection request to complete a connection.
X$TRAN	Transmits a message.
X$RCV	Receives a message.
X$CLR X$FCLR	Clears a virtual circuit connection.
X$UASN	Unassigns a port.
X$WAIT	Does a timed wait for the next network event completion.

X$CLRA	Clears all active virtual circuits and unassigns all ports.
X$GVVC XLGVVC	Passes control of a virtual circuit to another application process.
X$STAT	Returns various status information related to PRIMENET.

Note

In this and all succeeding syntax descriptions, each argument is defined as a two-byte integer or array of two-byte integers. The FTN definition of a 4-byte integer is INTEGER*2; the PL/I equivalent is FIXED BIN (15). The F77 definition of a 4-byte integer is INTEGER*4; the PL/I equivalent is FIXED BIN (31).

SUBROUTINE DESCRIPTIONS

The subroutines used to make a virtual call are grouped functionally in order of appearance. The remaining routines follow in alphabetical order.

Assigning a PRIMENET Port to Receive Incoming Call(s)

▶ X$ASGN

X$ASGN assigns a port. To receive incoming network connection requests, an application must assign one or more of the available network ports. The call to X$ASGN instructs PRIMENET to direct each connection request that specifies a port to the connection request queue of the assigning process. The process may read this queue by calls to the X$GCON or XLGCON subroutines.

The only time PRIMENET does not assign a legal port to a process is when there is not enough buffer space. PRIMENET puts multiple requests for the same port in a first-in-first-out queue. When the process at the head of the queue unassigns that port (see X$UASN), the next process waiting in queue for that port begins receiving incoming connection requests.

The assigning process may request automatic deassignment of a particular port after a specified number of connection requests are directed to it. The count field in the X$ASGN call is used to specify this. If count is a positive integer, PRIMENET will remove the assigning process from the port after processing count connection requests for that port. The assigning process must assign the port again to reenter the queue. A count of 0 prevents automatic deassignment.

A count of -1 causes the assignment request to be placed and kept at the back of the assignment queue for the specified port. PRIMENET places ahead of this request in the queue any process with a non-negative count that calls X$ASGN. A process with a negative count will handle a request only when no other processes are available. It returns to the bottom of the queue when a process with a non-negative count assigns the specified port.

If there is more than one process with a count of -1 assigning the same port, only the first will handle bottom-of-queue requests, until it unassigns itself or logs out. Then, the next negative count process will assume the bottom-of-queue position.

For example, several server processes, each with a count of 1, assign one port. Each process handles a single request for service, and is immediately deassigned, allowing the next incoming request to be taken by the next server in the queue. An error-handling process is given a count of -1 and sits at the bottom of the queue. When no server process is available, the error handler takes the incoming requests.

If one process makes two successive calls to X$ASGN with the same port, the count of the queued first assignment request call is replaced by the count of the second. The position of the request in the assignment queue remains unchanged.

Call syntax:

```
CALL X$ASGN(port, count, status)
```

Arguments:

port — INTEGER*2. The port to be assigned, 1 to 99 inclusive.

count — Integer*2. The number of incoming requests to be directed to the assigning process before automatic deassignment. (See text above.)

'count' > 0: Automatic unassignment after 'count' calls.
'count' = 0: Infinite assignment.
'count' < 0: Remain at end of assignment queue.

status — INTEGER*2. Returned. The returned status of the call.

The following is a list of the status codes that may be returned by a call to X$ASGN.

XS$BPM — The port specified in the call is not in the legal range of 1 to 99.

XS$CMP The assign request has been successfully placed at the front of the assign queue for the specified port.

XS$MEM No system resources are currently available for more assign requests.

XS$NET Networks are not configured for this system.

XS$QUE The assign request is behind at least one other request for this port.

Call Requesting

▶ X$CONN -- X$FCON -- XLCONN

Any of the call requesting subroutines initiates the establishment of a virtual circuit. The application supplies a virtual circuit status array (as the vcstat argument). The result of the call request will be returned into this virtual circuit status array. Normally, this call request status will change with time, as the call progresses. The call request subroutine will also return a VCID, to be used with all subsequent IPCF subroutine calls related to this virtual circuit.

The caller must specify the target node and a port there to make a call request. For short form routines, only configured node names can be used. The long form routine (XLCONN), however, permits numeric addresses as well as node names.

X$CONN is the short form routine, intended to initiate a connection to another PRIMENET application that is executing on any other Prime node in the network.

X$FCON is the short form for fast select. It adds the capability of sending the call user data field, and retrieving called user or clear user data returned by the callee.

XLCONN allows you to specify in detail each of the fields in a call request packet. In addition, you can put some constraints on which network path to use, and retrieve returned user data fields.

Note

When PRIMENET is connected to a public data network, the agency controlling that network may impose restrictions on the use of the fields in the call request packet.

Refer to Appendix A for more information on X.25 facility fields.

Call syntax:

```
CALL X$CONN(vcid, port, tadr, tadrn, vcstat)

CALL X$FCON(key, anskey, vcid, port, tadr, tadrn, udata, udatan,
            vcstat, rudat, rudatn, rudabc)

CALL XLCONN(key, vcid, port, tadr, tadrn, fcty, fctyn, prid, pridn,
            udata, udatan, vcstat, [rudat, rudatn, rudabc])
```

Arguments:

key — INTEGER*2. Describes the form of the call and the physical paths to be allowed for the connection. The key has three additive parts. They are selected from:

Address format,

either:

XK$NAM	tadr contains an ASCII PRIMENET node name (default, so XK$NAM may be omitted)

or:

XK$ADR	tadr contains the ASCII subscriber address

Path specification,

XK$ANY	Any available network path OK
XK$PDN	Path through packet network OK
XK$RNG	RINGNET (PNC) path OK
XK$RTE	Route-through path OK
XK$SYN	Synchronous link path OK

Note

At least one path specifier must be present in the call.

Facility option request (used with XLCONN only),

XK$FCT — A default facilities field will be provided by PRIMENET, appropriate for the connection type (RING or specific PDN), specified separately by the application, or not used. (It might be added by your PDN.)

Returned data option (used with XLCONN only),

XK$RTD — This key indicates that the optional return data arguments for the fast select facility have been supplied.

anskey — INTEGER*2. For X$FCON, selects restricted response or not for fast select calls.

XK$ACC — The callee may accept or clear the call;

XK$CLR — The callee must clear the call.

vcid — INTEGER*2. Returned. The VCID to be used for this connection. Of no interest, with X$FCON, when the XK$CLR anskey is used.

port — INTEGER*2. The port assigned by the process that is the target for this connection request. (Ignored if the prid argument is used.)

tadr — Array. It holds a string of bytes (char nonvarying in PL/I). This array contains the PRIMENET node name of the target node (with a maximum of 6 characters). For XLCONN, combined with XK$ADR, it holds the target node's address. If you are calling by address, the maximum length is 15 characters.

tadrn — INTEGER*2. The number of characters in tadr.

fcty — Array. Contains the bytes to go into the call request packet facilities field. (Ignored if fctyn is 0.)

fctyn — INTEGER*2. The number of bytes to be taken from fcty. Legal range is 0 to 63. (Must be 0 if XK$FCT is used.)

prid — Array. A buffer that contains the four bytes to go into the protocol id field call request packet. (Ignored if pridn is 0.)

pridn — INTEGER*2. The number of bytes to be taken from prid. Legal values are 0 and 4.

Note

If pridn is 0, PRIMENET uses the protocol id field to pass the port specified in port. In this case, this field is used for a host-to-host protocol format defined for PRIMENET.

If pridn is 4, the application supplied bytes are used. In this case, the value specified in port is not used.

udata — Array. A buffer that holds the bytes to go into the user data field call request packet. (Ignored if udatan is 0.)

udatan — INTEGER*2. The number of bytes to be taken from udata. Legal values are 0 to 12, except for fast select calls, for which the range is 0 to 124.

vcstat — Two INTEGER*2 word array. Returned. Used for the virtual circuit status array. The list of returned virtual circuit status codes follows separately after the argument descriptions.

The vcstat may be written into during the whole life of the virtual circuit. Ensure it is the appropriate type.

For X$FCON (Optional), the following three arguments are mandatory for X$FCON and optional for XLCONN:

rudat — Array. To hold a string of bytes. Returned. This array will receive returned user data fields from call accept and clear packets, if present. It is intended for use with fast select calls. Note that the entire X.25 user data field will be returned here (refer to Appendix A).

rudatn — INTEGER*2. The maximum number of characters to be put into rudat.

rudabc — INTEGER*2. Returned. The actual number of characters returned into rudat.

Note

rudat and rudabc are only valid when the call request has completed, that is when vcstat(1) equals either XS$CMP or XS$CLR.

The following list of codes can be returned into the first word of the virtual circuit status array vcstat. These codes could be expected immediately on return from the call request subroutine.

XS$BPM — One of the arguments to the call is missing, out of range, or in conflict with other arguments.

XS$DWN — The target node specified in tadr is currently unavailable through PRIMENET.

XS$FCT — Bad facility field supplied (XLCONN only).

XS$IP — The connection request (or data transfer) has been successfully initiated. See XS$CMP.

XS$MAX — This request exceeds the maximum number of virtual circuits allowed for the physical node-to-node link. With normal PRIMENET configurations, this will occur only if the link includes a PDN, which may only allow relatively few simultaneous virtual circuits.

XS$MEM — There is temporary buffer congestion in the local network. The system currently does not have the resources required to process the request. Retry the request later on. (X$FCLR only. X$CLR automatically retries the clear request later.)

XS$NET — PRIMENET is not configured on this Prime system.

XS$UNK — The target node specified in tadr is unknown in the network (not configured).

As a result of data transfers (network and remote user actions), the following codes might be returned later.

XS$CLR — The connection has been cleared and is no longer usable. If the connection was cleared by the network or the remote process (not the local process), the second word of the virtual circuit status array will hold the clearing cause and diagnostic code. (Refer to Chapter 11.)

XS$CMP — The connection attempt or a data transfer has successfully completed.

XS$MAX — This request exceeds the maximum number of virtual circuits allowed for the physical node-to-node link. With normal PRIMENET configurations, this will only occur if the link includes a PDN, which may only allow relatively few simultaneous virtual circuits.

XS$RST The virtual circuit has been reset. All operations in progress have been aborted.

Find Information on Incoming Call

▶ X$GCON -- X$FGCN -- XLGCON

These subroutines return information about incoming call requests. Once an application has assigned a port, PRIMENET places all incoming call requests that specify that port on a call request queue for that process. Because each PRIMENET application has only one such queue, when a process has several ports assigned, the connection requests for each of them are placed on this same queue in a first-in-first-out fashion.

A call to a 'get information on call request' subroutine copies information about the first call request on the process's queue without dropping the request from the queue. The process should then dispose of the pending connection by either accepting it or clearing the call. Either action drops the pending request from the call request queue. Requests not handled by the process in a timely fashion are automatically cleared by PRIMENET. This timeout is 100 seconds.

X$GCON is the short form subroutine, primarily for use between Prime nodes.

X$FGCN is the short form subroutine for retrieving information for fast select calls. It gives the name of of the caller's node, and the call user data field. It also tells if the caller required restricted response or not. (Fast select calls with restricted response required must be cleared.)

XLGCON allows the caller to extract almost all of the fields in an X.25 call request packet.

Call syntax:

```
CALL X$GCON(vcid, port, status)

CALL X$FGCN(key, anskey, vcid, port, fadr, fadrn, fadrbc,
            udata, udatan, udatbc, status)

CALL XLGCON(key, vcid, port, fadr, fadrn, fadrbc, fcty, fctyn,
            fctybc, prid, pridn, pridbc, udata, udatan,
            udatbc, status)
```

Arguments:

key — INTEGER*2. X$FGCN and XLGCON: Describes the format of the calling node name to be placed into fadr.

- XK$NAM — fadr will receive the ASCII PRIMENET node name.
- XK$ADR — fadr will receive the ASCII subscriber address.

anskey — INTEGER*2. Returned. For X$FGCN, Indicates restricted response or not for fast select calls.

- XS$ACC — The callee may accept or clear the call.
- XS$CLR — The callee must clear the call.
- XS$NOT — This is not a fast select call.

vcid — INTEGER*2. Returned. This vcid is to be used for all subsequent IPCF calls, relating to this virtual circuit.

port — INTEGER*2. Returned. The port to which this call request is directed.

fadr — Array. It holds a string of bytes — char nonvarying. Returned. This buffer will contain either the PRIMENET node name or the system address for the node at which this call originated.

fadrn — INTEGER*2. The maximum number of bytes fadr may receive.

fadrbc — INTEGER*2. Returned. The number of bytes returned into fadr.

fcty — Array. Returned. It holds a string of bytes — nchar nonvarying. The buffer that will receive a copy of the call request packet facilities field. (Ignored if fctyn is 0.)

fctyn — INTEGER*2. The maximum number of bytes the buffer named by fcty may receive.

fctybc — INTEGER*2. Returned. The number of bytes returned into fcty by call.

prid — Array. Returned. It holds a string of bytes — char nonvarying. The buffer that will receive the bytes from the call request packet protocol id field. (Ignored if pridn is 0.)

pridn — INTEGER*2. The maximum number of bytes prid may receive.

pridbc — INTEGER*2. Returned. The number of bytes returned into prid by call.

udata — Array. Returned. It holds a string of bytes — char nonvarying. The buffer that will receive the bytes from the user data field of the call request packet. (Ignored if udatan is 0.)

udatan — INTEGER*2. The maximum number of bytes udata may receive.

udatbc — INTEGER*2. Returned. The number of bytes returned into udata by call.

status — Two-word array, INTEGER*2. Returned. Contains the returned status of the call.

The following is a list of status codes that may be returned in the first status word.

XS$NOP — No call requests pending.

XS$CMP — Pending call request: return arguments are valid.

XS$BPM — Invalid key argument in the call.

XS$NET — Networks not configured.

The second status word is valid only when the first word has the value XS$CMP. The second word may have a value of either 1 or 2. These codes are defined below.

1 — A new incoming call request

2 — A "passed off" virtual circuit; see X$GVVC/XLGVVC

Call Acceptance

▶ X$ACPT -- X$FACP -- XLACPT

Any of the call accept subroutines can be used to accept a connection request and complete the connection. The application that has first identified a caller through any of the 'get connection data' subroutines (X$GCON -- X$FGCN -- XLGCON -- XLGC$) accepts the connection or clears it. The status of the call is returned in the vcstat array, given as argument. Furthermore, this virtual circuit status array is used throughout the life of the virtual circuit.

X$ACPT is the short form subroutine, primarily for use between Prime nodes.

X$FACP is the short form subroutine for fast select call accept, for transfer of returned user data within the call accept packet.

XLACPT allows you to specify in detail the X.25 packet-level protocol call accept packet. This long form may contain any (or all) of facilities, protocol id, and user data field.

Note

When PRIMENET is connected to a public data network, the agency controlling that network may impose restrictions on the use of the fields in the call accept packet.

The X.25 standard only permits called user data on accepts of fast select calls.

Call syntax:

```
CALL X$ACPT (vcid, vcstat)

CALL X$FACP(vcid, udata, udatan, vcstat)

CALL XLACP(key, vcid, fcty, fctyn, prid, pridn,
           udata, udatan, vcstat)
```

Arguments:

key — INTEGER*2. Either 0, for a user-specified facilities field, or XK$FCT for a PRIMENET-supplied facility field appropriate to the particular circuit (RING or specific PDN).

vcid — INTEGER*2. The virtual circuit id for this circuit. This value was obtained by a preceding call to X$GCON, X$FGCN, or XLGCON.

fcty — Array. Contains the bytes to go into the call accept packet facilities field. (Ignored if fctyn is 0.) Must not be used when key is set to XK$FCT.

fctyn — INTEGER*2. The number of bytes to be taken from fcty. Legal range is 0 to 63. (Must be 0 if XK$FCT is used.)

prid — Array. A buffer that contains the four bytes to go into the call accept packet protocol id field. (Ignored if pridn is 0.) If udata follows and is not supplied by the application, PRIMENET will set its default format.

pridn — INTEGER*2. The number of bytes to be taken from prid. Legal values are 0 and 4. (The value should be 0 for non-fast-select accepts through a PDN.)

udata — Array. A buffer that holds the bytes to go into the user data field of the call accept packet. (Ignored if udatan is 0.)

udatan — INTEGER*2. The number of bytes to be taken from udata. Legal values are 0 to 12, except for accepts on fast select calls, when the range is 0 to 124. (The value should be 0 for non-fast-select accepts through a PDN.)

vcstat — Two word array. INTEGER*2. Returned. Used for the virtual circuit status array.

Note that vcstat may be written into during the whole life of the virtual circuit. Ensure it is of appropriate type for this.

The following is a list of status codes that may be returned in the first word of vcstat.

XS$BPM — Invalid arguments in the call (not for X$ACPT).

XS$BVC — The calling process does not control the virtual circuit specified by vcid.

XS$FCT — Bad facility field supplied (XLACPT only).

XS$IDL — The operation was successful and the virtual circuit is now idle, awaiting data traffic.

XS$ILL — The process tried to ACCEPT a virtual circuit that was not in the call request pending state, or tried to accept a call set up for fast select — restricted response.

XS$MEM — There is temporary buffer congestion in the local PRIMENET node. Retry the accept several seconds later.

As a result of data transfers, actions in the network, or remote user actions, the following codes might be returned.

XS$CLR	The connection has been cleared. The second word of the virtual circuit is now the valid clearing cause.
XS$CMP	A data transfer operation has completed successfully.
XS$RST	The virtual circuit has been reset. All operations in progress have been aborted.

Transmit Data

▶ X$TRAN

X$TRAN is the PRIMENET transmit-message subroutine. An application calls it to send the contents of a buffer through the network to the process on the opposite end of a virtual circuit. PRIMENET automatically splits the message into X.25 packets of appropriate size for transmission, and then recombines them at the receiving side.

PRIMENET supports X.25's two data levels and interrupt procedure. When applications call X$TRAN, they supply an argument level with one of three values (the SYSCOM>X$KEYS.INS.XXX files have defined mnemonics for each value) to indicate one of the following:

- A message (data packet sequence): Q-bit set to 0 (XT$LV0)
- A message (data packet sequence): Q-bit set to 1 (XT$LV1)
- An interrupt packet (XT$INT)

Both XT$LV0 and XT$LV1 are requests to move up to 32,767 (32K) bytes of data. The only difference between them is their data level. PRIMENET passes the data level transparently through the circuit to the receiver so that an application may distinguish between data messages with the Q-bit values set differently. PRIMENET treats XT$LV0 and XT$LV1 data packets the same, handling data transmission requests of both types in a single queue in first-in-first-out fashion.

Interrupt packets (XT$LV2) are handled separately, corresponding to the X.25 packet-level protocol. Each can carry only a single byte of data, which is placed at the top of the queue ahead of all ordinary data packets. As a result, an interrupt packet may arrive at its destination earlier than normal data sent before it. For the effect of interrupts on the receiving side, see X$RCV.

Call syntax:

```
CALL X$TRAN(vcid, level, buffer, bufbc, status)
```

Arguments:

vcid — INTEGER*2. The VCID for this circuit.

level — INTEGER*2. The data level of this message is:

XT$LV0 (0) for normal data packets with the X.25 Q-bit set to 0.

XT$LV1 (1) for normal data packets with the X.25 Q-bit set to 1.

XT$INT (2) for an interrupt packet.

buffer — Any array. The data buffer to be moved through the virtual circuit. buffer must not cross a segment boundary.

bufbc — INTEGER*2. The number of bytes to copy from buffer. bufbc is 0 to 32767 except for interrupt packets. For interrupt packets, bufbc is 0 or 1.

status — INTEGER*2. Returned. The status of this transmit.

Codes that occur immediately on return from X$TRAN appear below.

XS$BPM — There are invalid arguments in the call.

XS$BVC — The calling process does not control the virtual circuit specified in vcid.

XS$ILL — The transmit operation is illegal because a circuit connection request or a clear request is pending. This is the result of attempting transmission over an "almost-open" or "almost-closed" circuit.

XS$IP — The transmit is in progress. status will be further updated by the completion or failure of the operation. (This is the normal immediate return code from X$TRAN.)

XS$MAX — This request exceeds the maximum number of transmits that can be in progress simultaneously over a single virtual circuit. This request to initiate another transmission is denied.

XS$MEM — There is temporary PRIMENET buffer congestion on your local node that prevents the acceptance of the transmit request at this time.

The codes that may be returned in status later appear below.

XS$CLR — The virtual circuit has been cleared. See the virtual circuit status array for the clearing cause and diagnostic code.

XS$CMP — The transmit is complete. The message has been copied out of the sender's buffer and transmission is initiated. (A transmit status of complete means only that PRIMENET will attempt to deliver the message. Applications requiring assured delivery must implement their own end-to-end acknowledgement in a higher level protocol of their own.)

XS$RST — The virtual circuit has been reset. The status of this transmit request is unknown and no further attempts will be made to complete it.

Receive Data

▶ X$RCV

X$RCV is the PRIMENET receive-message subroutine. An application offers a buffer into which PRIMENET places received messages from the specified virtual circuit. The application receiving a message should establish its receive buffer before the other application attempts to transmit data.

In all cases, data messages transmitted through the X$TRAN subroutine are reproduced identically in the receive buffer. However, several cases of mismatched buffer sizes and interrupt handling are worthy of note.

The simplest case is a receive buffer that is at least big enough (the same size or bigger) to contain an incoming data message. In such a case, the receive status is set to indicate a completed receive as soon as the entire message is copied into the receive buffer.

When the receive buffer is too small to contain an incoming message, the specified buffer is filled, the receive status is set to indicate a completed operation, and the remainder of the message is held until another buffer, presented by a call to X$RCV, is available. PRIMENET attempts to complete the delivery of the message; if necessary, it fills the second buffer and again holds the remainder. This will continue until the complete message is copied into the receiver's buffers.

A process is allowed to send an interrupt across a virtual circuit even when regular data transmissions are in progress. (See X$TRAN.) Because an interrupt may pass normal data moving in the network, a partially completed receive may be interrupted by such an interrupt

message. An application receives interrupts in the same way it receives regular data with X$RCV.

However, if a receive buffer offered in a call to X$RCV is partially filled with level 0 or level 1 data when an interrupt message is received, the following actions take place. The status word of the X$RCV request that is currently being filled is marked as completed, without waiting for the receive buffer to be completely filled. The next pending call to X$RCV serves to receive the interrupt, and the call to X$RCV after that receives the remainder of the original message. No data loss results in this exchange, but the original data message is broken into two pieces. The caller should check the returned status word to see what it is.

Call syntax:

```
CALL X$RCV(vcid,buffer,bufn,status)
```

Arguments:

vcid — INTEGER*2. The VCID for this circuit.

buffer — Any array. Returned. The data buffer into which incoming data should be moved. buffer must not cross a segment boundary.

bufn — INTEGER*2. The maximum number of bytes that may be moved into buffer.

status — Three-word array, Integer. Returned.

The first word is the receive request status word.

The second word is set to the level of the incoming data.

The third word is set to the number of bytes moved into buffer.

The codes that may be returned immediately in the first word of status on return from X$RCV appear below.

XS$BPM — Invalid arguments in the call.

XS$BVC — The calling process does not control the virtual circuit specified in vcid.

XS$ILL The receive operation is illegal because a circuit connection request or a clear request is pending. This is the result of putting up a receive on an "almost-open" or "almost-closed" circuit.

XS$IP The receive is in progress. status will be further updated by the completion or failure of the operation. (This is the normal immediate return code from X$RCV.)

XS$MAX This request exceeds the maximum number of receives that can be in progress simultaneously over a single virtual circuit. This request to initiate another receive is denied.

XS$MEM There is temporary PRIMENET buffer congestion on your local node that prevents the acceptance of the receive request at this time.

Codes that may be returned later appear below.

XS$CLR The virtual circuit has been cleared. See the virtual circuit status array for the clearing cause.

XS$CMP The receive is complete. The incoming data have been moved to buffer, and the second and third words of status are updated.

XS$RST The virtual circuit has been reset. The status of this operation is unknown and no further attempts will be made to complete it.

Clear

▶ X$CLR — X$FCLR

These subroutines disconnect a virtual circuit by initiating transmission of a clear request. At any time during the life of a virtual circuit, an application may clear (break) the circuit. First, the VCID must be passed to the application by a connect call or a request for information concerning pending incoming calls. Then, a call to XCLR/XFCLR for that VCID cancels all activities in progress, releases any resources, and notifies the process on the other end of the circuit that a clear has been requested. Fast select call requests may be cleared, including return of a clear user data field, immediately after the call request has been received.

The actual clearing process happens in two stages. First, the call to the 'clear' subroutine initiates transmission of a clear request through PRIMENET to the other end of the virtual circuit. To indicate this, the clear subroutine call returns a status code immediately. If the call is successful, all transmit and receive requests in progress are immediately aborted, which is indicated by their status being changed to XS$CLR. Secondly, when the remote system receives the clear request, it answers by transmitting back a clear confirmation message. The clear subroutine call returns a status code immediately. If the call is successful, all transmit and receive requests in progress are immediately aborted. Thus, only when the clear confirmation has been returned successfully does the clear requesting caller see the circuit as cleared in the first word of his virtual circuit status array (vcstat). The second word of the virtual circuit status array is invalid in this case. (If the clear is requested before the virtual circuit was accepted, there is no virtual circuit array defined; therefore, the application cannot detect the confirmation.)

When the remote process initiates the clear request, the local PRIMENET process (NETMAN) automatically sends the clear confirmation, aborts all data transfer operations in progress, and sets the first word of the virtual circuit status array to the 'circuit cleared' value (XS$CLR). (Chapter 11 describes the clearing codes.) The second word is set to the valid clearing cause and diagnostic code.

Note

The delay for the clear confirm might be noticeable, especially over long-distance PDN links. Suppose the application that requests a clear returns to PRIMOS (by a 'CALL EXIT') before the XS$CLR status has been returned into the virtual circuit status array, and the application then executes another process. In such a case, there is a risk that the latter program will be overwritten by PRIMENET when the clear confirm arrives. To avoid this, the application should call X$CLRA to immediately drop all its virtual circuit references before returning to PRIMOS command level.

X$CLR is the short form routine, primarily intended for use between Prime nodes. X$FCLR is specifically intended for the clearing of fast select calls, when the application wants to return clear user data (XT$LV0, XT$LT1, or XT$INT). (X.25 permits this only as an immediate response to a fast select call request.)

Note

The clear user data field's value is given by a single argument, clrudat, that corresponds to the concatenated arguments prid and udata of the X$FACP/XLACPT routines. (See Appendix A.)

Call syntax:

CALL X$CLR(vcid, why, status)

CALL X$FCLR(vcid, why, clrudat, clrudatn, status)

Arguments:

vcid	INTEGER*2. The VCID of the circuit to be cleared.
why	INTEGER*2. The low-order byte of why is taken as the diagnostic code. (Refer to Chapter 11 for information on diagnostic codes.)
clrudat	Array. A buffer that holds the bytes to go into the user data field of the clear request packet. (Ignored if clrudatn is 0.)
clrudatn	INTEGER*2. The number of bytes to be taken from clrudat. The legal value is 0, except following fast select calls, when the range is 0 to 128.
status	INTEGER*2. Returned. It contains the immediate return status of the call.

The following is a list of the status codes that may be returned.

XS$BPM	One of the arguments to the call has an illegal value.
XS$BVC	The calling process does not control the virtual circuit specified by the vcid.
XS$CMP	The operation is successful. All pending transmits and receives are aborted with a status of XS$CLR.
XS$ILL	A clear with user data has been requested for a fast-select virtual circuit, but it does not immediately follow the call request.
XS$MEM	There is temporary buffer congestion in the local network. The system currently does not have the resources required to process the request. Retry the request later on. (Necessary with X$FCLR only, because X$CLR uses a built-in memory mechanism to initiate the clear request later. Refer to X$STAT, virtual circuit status list.)

Release a PRIMENET Port

▶ X$UASN

X$UASN unassigns a port. At any time, an application may ask to be removed from the assignment queues for one or all of the ports currently assigned by it. The application's assign request for the port specified in the call is immediately deleted from the assignment queue regardless of its position in the queue.

If the value of the specified port is < 0, all of the application's port assignment requests are dropped from the assignment queues.

This operation always completes successfully. If the port passed in the call is not assigned at the time of the X$UASN call, no action is taken.

Call syntax:

```
CALL X$UASN(port)
```

Argument:

port	INTEGER*2. The port to be unassigned. (< 0 means all ports.)

General Network Cleanup

▶ X$CLRA

X$CLRA clears all active virtual circuits, reinitializing an application's network environment. Any pending network operations are aborted and all of the virtual circuits held by the application are cleared.

Note

Unlike XCLR, XCLRA does not wait for confirmation from the application on the other side of the circuit before marking it cleared. Therefore, the virtual circuit status word of a circuit cleared by a call to X$CLRA is never updated to show that the circuit is cleared.

In addition to clearing all open virtual circuits, X$CLRA unassigns a process's interest in all ports. In this regard, it is equivalent to the call: X$UASN (-1).

This subroutine also drains the application's network (X$WAIT) semaphore, reducing the chances for spurious network event signals. See the Subroutines Reference Guide for complete information on semaphores.

Call syntax:

```
CALL X$CLRA
```

The operation always completes successfully. If no virtual circuits are open, or no ports are assigned, then no action is taken. It is good programming practice to use X$CLRA to clear each circuit and wait for clear confirmation before exiting any process or subsystem within an application.

Wait for Completed PRIMENET Action

▶ X$WAIT

X$WAIT does a semaphore wait for a network activity to complete. Optionally, this wait can be combined with a finite time-out period. Most of the IPCF subroutines initiate an activity and then immediately return, to allow the application to continue processing while the requested network action completes. The X$WAIT call provides a mechanism by which applications can ask to have processing suspended until any of their network actions completes.

When treated as an INTEGER*2 FUNCTION, X$WAIT returns a code value that indicates if the cause of the resumption of execution was a completed PRIMENET action or time-out.

A network activity is considered complete whenever the status of the corresponding request indicates that PRIMENET will take no further action on that request. In general, this includes any return status except the operation-in-progress code (XS$IP). (A code of XS$IP is always updated by PRIMENET as soon as the relevant activity completes.) A suspended process is also awakened when PRIMENET receives a connection request for that process.

Note

X$WAIT is implemented as a PRIMOS-quittable semaphore. As such, a suspended process may be awakened even though the action has not completed. Code using X$WAIT should make provision for this.

The network semaphore collects network events into a combined single notification when the application is not waiting on its network semaphore, to prevent event count rollover. An application should be aware that when it is awakened by X$WAIT, it can have multiple network activities completed. In that case, a new X$WAIT call that anticipates one of those activities to complete will not cause a wake up until yet another request completes, and an infinite hang could occur. Accordingly, applications should test all outstanding requests when executing. (For further details, refer to Chapter 15, IPCF Programming Examples, and Chapter 16, IPCF Programming Strategy.)

Call syntax:

```
CALL X$WAIT(time)

code = X$WAIT(time)
```

Arguments:

time — INTEGER*2. The number of tenths of seconds to remain suspended if no network action completes. (If time is 0, wait indefinitely.)

code — INTEGER*2. Returned function value. May be 0 or 1, as shown below.

- 0 Some network action (not necessarily the awaited one) completed before the timer expired.
- 1 The timer may have expired before any network action completed.

Virtual Call Transfers

▶ X$GVVC -- XLGVVC

X$GVVC/XLGVVC passes control of a virtual circuit to another process on the same PRIMENET node. The process issuing the call relinquishes the right to clear and to send or receive data through the specified circuit. The circuit is placed in the connection request pending queue for the target process, and is treated thereafter in the same manner as an incoming connection request.

Calls can be passed to a specific application by its user number, or to the owner of a PRIMENET port.

If the original call request/call accept packet has been released, the application can generate a simulated call request packet to handle passing control of the virtual circuit. This feature could be used for information transfer to the target application, conveyed in the user data field. (Call request packets are released when the virtual circuit is accepted; call accept packets are released by the first transmit/receive request on the circuit.)

An application can pass control of a circuit in the incoming request queue without accepting it. That leaves the initial choice of accepting or clearing the circuit to another application. In that case, the process to which the circuit is being passed sees the circuit connection request as a new request, not as one being passed.

Control of a circuit cannot be passed under the following conditions.

1. The application issued the call to X$CONN, X$FCON, or XLCONN to create this circuit and the circuit establishment has not yet been completed (that is, the virtual circuit status has not yet been set to XS$CMP).

2. The application wishing to pass control of the circuit has an in-progress call to X$RCV or X$TRAN.

3. The virtual circuit is in the process of being passed off.

4. The virtual circuit is a remote login circuit.

5. There are conflicts between the contents of the existing call request/accept packet and the user-requested call request packet.

Note

While being passed off, a virtual circuit has no owner, and thus no real virtual circuit status array. A scratch array within PRIMENET is temporarily used. The implication is that any virtual circuit status changes during this time are not transferred to an application's virtual circuit status array.

The target application requests and receives information about passed-off virtual circuits by a call to any of the 'get connection data' subroutines (X$GCON and others). Either of these will return the VCID, by which the target application references the connection being passed. Just as with new incoming connection requests, pass-off connection requests must be cleared or "accepted" (X$ACPT, XLACPT) before being used for data transfer. As with new incoming connection requests, a passed connection that has not been accepted or explicitly cleared within 90 seconds after it enters the incoming request queue is automatically cleared by PRIMENET.

X$GVVC is the short form routine, limited to pass off a call by target process user number. XLGVVC is the long form, which also permits passoff by the port and possibly supplies call request data for the passoff.

Call syntax:

```
CALL X$GVVC(vcid, userno, status)

CALL XLGVVC(key, vcid, userno,
            port, fadr, fadrn, tadr,
            tadrn, fcty, fctyn, prid, pridn,
            udata, udatan, status)
```

Arguments:

key — INTEGER*2. Specifies selection of target process.

- XK$USR — pass off to user number userno
- XK$PRT — pass off by PRIMENET port port

vcid — INTEGER*2. The VCID for the circuit being passed.

userno — INTEGER*2. The user process number of the process to which this circuit is being passed. Used only when key = XK$USR. The legal range is 2, up to and including the sum of the CONFIG directives NTUSR + NPUSR + NRUSR + NSLUSR.

port — INTEGER*2. The port number to be used to pass the virtual circuit. Used only when key = XK$PRT. Range is 1 - 99.

fadr — Array. It holds a string of bytes - char nonvarying. This array contains the address of the simulated call originating network node.

fadrn — INTEGER*2. The number of characters in fadr.

tadr — Array. It holds a string of bytes - char nonvarying. This array contains the node address of the target node. For PRIMENET nodes, the maximum length is 6. If you are using an address, the maximum is 15.

tadrn — INTEGER*2. The number of characters in tadr.

fcty — Array, holding a string of bytes. Contains the bytes to go into the simulated call request packet facilities field. (Ignored if fctyn is 0.)

fctyn INTEGER*2. The number of bytes to be taken from fcty. Legal range is 0 to 63.

Note

In contrast to initial call requesting and acceptance, there is no check done for legality of the supplied facility field. Also, the facilities requested by this facility field are completely ignored. The virtual circuit's 'parameters' remain unchanged. It is suggested that the facility field from the virtual circuit's creation be copied, if the application wants to transmit a facility field.

prid Array. A buffer that contains the four bytes to go into the simulated call request packet protocol id field. (Ignored if pridn is 0.)

pridn INTEGER*2. The number of bytes to be taken from prid. Legal values are 0 and 4.

Note

If pridn is 0, PRIMENET uses the protocol id field to pass the port specified in port. In this case, this field is used for a host-to-host protocol format defined for PRIMENET.

If pridn is 4, the application supplied bytes are used. In this case, the value specified in port will still control the virtual circuit.

udata Array. A buffer that holds the bytes to go into the user data field of the call request packet. (Ignored if udatan is 0.)

udatan INTEGER*2. The number of bytes to be taken from udata. Legal values are 0 to 124. It should be 0 for all non-fast-select accepts.

status INTEGER*2. Returned. Contains the return status of the call.

The following is a list of the status codes that may be returned by a call to X$GVVC.

XS$BVC The calling process does not control the virtual circuit specified by vcid.

XS$BPM One of the arguments to the call is missing, out of range, or in conflict with other arguments.

XS$CMP The operation was successful. This virtual circuit is now pending on the target process's connection request queue.

XS$ILL This virtual circuit is in one of the states described above during which pass-off is prohibited.

XS$MEM There is temporary buffer congestion in the local PRIMENET. The system currently does not have the resources required to process the request. Retry the request later on.

XS$UNK The target application is not logged in or the call cannot be passed by use of the specified port.

Network Status Interrogations

▶ X$STAT

X$STAT may be called at any time to determine the state of the network. The value given in key specifies the type of status information to be returned. X$STAT returns information about the local system's PRIMENET configuration, the currently open virtual circuits, and the mapping of ASCII PRIMENET names to their X.25 addressing form equivalents. The parameters num, array1, alen1, array2, and alen2 are input arguments, returned values, or unused, depending on the value of key.

Call syntax:

```
CALL X$STAT(key,num,array1,alen1,array2,alen2,code,time)
```

Arguments:

key INTEGER*2. Specifies information to be returned.

XI$ADR Returns all X.25 addresses in network.

XI$AVC Returns VCIDs of all circuits that are open to or from a specific X.25 address.

XI$VCD Returns information about a specific virtual circuit.

XI$XTP Returns the PRIMENET name of an X.25 address.

XI$PTX Returns the X.25 address of a PRIMENET name.

XI$MYN Returns the X.25 address and PRIMENET name of the application's system.

XI$PDN Returns names of all accessible public data networks.

XI$PVC Returns VCIDs of all circuits that are open to or from a specific public data network.

XI$RLG Returns the VCID and remote address of a process's remote login circuit.

num INTEGER*2. Conditionally returned. The number of network addresses, number of virtual circuits, number of accessible public data networks or no meaning (depending on key).

array1 Array. Conditionally returned. If defined by INTEGER*2 words, it should be dimensioned to the size of the configuration loaded on the node. A buffer containing X.25 addresses, public data network names, or PRIMENET names (in ASCII, with two characters per array entry).

alen1 INTEGER*2. Conditionally returned. It indicates the "actually used" length of array1.

array2 Array. Conditionally returned. If defined by INTEGER*2 words, it should be dimensioned to at least 256 words. A buffer containing VC identifiers, VC status information, the number of characters in each X.25 address, the number of characters in the PRIMENET system name, or the number of characters in the PDN name.

WARNING

At PRIMENET Rev. 19.3, it may be necessary to increase the lengths of the two return arrays array1 and array2 to cope with large network configurations. As a result, old programs risk being overwritten when PRIMENET needs to use more of these arrays than previously dimensioned. Users should review their applications that use X$STAT.

alen2 INTEGER*2. Conditionally returned. It indicates the "actually used" length of array2.

code INTEGER*2. Returned. It indicates outcome of call.

XS$CMP The operation was performed successfully.

XS$BPM Invalid arguments in the call.

XS$NET No network is configured.

XS$UNK The X.25 address, virtual circuit, PRIMENET name, or PDN name is unknown.

time Returned. INTEGER*2. The current time; retrieved in minutes since midnight.

Each type of status call is described below. The meanings of code and time are the same for each value of key. (Starred arguments (*) are input arguments, and the other arguments are returned by the call.)

XI$ADR num will contain the number of addresses in the network. array1 will contain the addresses, two characters per entry, one name right after the other. alen1 will contain the used length of array1. Each entry in array2 specifies the number of ASCII digits in the network addresses given in array1. alen2 will contain the number of used words in array2, which will equal the value of num.

To find the offset into array1 for a specific address, add the lengths of the previous addresses, converted into needed array words per address. Each address will use an integer number of array words, even if it has an odd number of digits.

All PRIMENET nodes have addresses, even if they are not connected to a PDN. In the latter case, PRIMENET provides a fictitious number calculated from the node name, starting with 9999. To find the node name corresponding to an address, call X$STAT using key XI$XTP.

XI$AVC num will contain the number of virtual circuits open to or from a specific network address. array1* specifies the address of interest. alen1* is the number of ASCII digits in the address of interest. The entries in array2 will contain the VCID's. alen2 will be set to the actual used length of array2, which will equal the value of num.

XI$MYN num has no meaning for this key and will not be modified. array1 will be set to the PRIMENET system name for this system. alen1 will contain the number of characters in this system's name. array2 will contain the X.25 address for "this" node. alen2 will contain the number of ASCII digits in the X.25 address.

XI$PDN num will contain the number of accessible packet data networks. array1 will contain these network names. alen1 will contain the used length of array1. Each entry in array2 will contain the number of ASCII characters per corresponding data network name. alen2 will contain the number of used words in array2, which will equal the value of num.

XI$PTX num has no meaning and will not be modified. array1* specifies the PRIMENET system name of interest. alen1* specifies the number of characters in array1. array2 will contain the X.25 address. alen2 will contain the number of ASCII digits in array2.

XI$PVC num will contain the number of virtual circuits open to the specified packet network. array1* will specify the packet data network (PDN) of interest. alen1* specifies the number of characters in the name of the PDN. The entries in array2 will be the circuit numbers. alen2 will be set to the actual used length of array2, which will equal the value of num.

XI$RLG num will be set to the VCID of the process's remote login circuit. array1 will contain the remote address, two characters per word. alen1 will be set to the length of array1, in characters. array2 and alen2 are not used.

XI$VCD num* specifies the VCID of interest. Each entry in array2 is described below. The length of array2 (alen2) is 13 words. array1 and alen1 are not used.

array2(1) Circuit status. See notes below.

array2(2) User process number.

array2(3) The maximum packet size in bytes.

array2(4) Packet window, maximum number of outstanding packets.

Note that the packet size and window size returned are the input direction sizes. Usually, the output direction sizes are the same, but X.25 permits facility negotiations that make them unequal. (If desired, the output sizes could be found by retrieving the input sizes at the

other end of the virtual circuit. If a PDN is part of the virtual circuit, the packet and window sizes might be changed when they pass through the PDN.)

array2(5) Port number of call.

array2(6) The number of resets since call began.

array2(7) Minutes open.

array2(8) First word of the number of packets received.

array2(9) Second word of the number of packets received.

(array2(8) concatenated with array2(9) thus forms an INTEGER*4 variable.)

array2(10) First word of the number of packets sent.

array2(11) Second word of the number of packets sent.

(array2(10) concatenated with array2(11) thus forms an INTEGER*4 variable.)

array2(12) Controller type. See notes below.

array2(13) PNC address or logical SMLC line number.

Values for circuit state and controller type are described below.

Circuit State	Meaning
1	Remote login (on this system)
2	Unused
3	Unused
4	Circuit being passed off
5	User data transfer
6	User local call request pending
7	User remote call request pending
8	User local clear request pending
9	Unused
10	Unused
11	Unused
12	Clear desired but no memory available. PRIMENET will automatically retry clear request
13	Unused

14	Remote log-through (placing a login to another node)
15	Has sent clear request; waiting for clear confirm

Controller Type	Meaning
1	IPC (Reserved)
2	SMLC
3	Ring (PNC)
4	Local connection within same machine

XI$XTP num has no meaning for this value of key and is not modified. array1* specifies the X.25 address of interest. alen1* specifies the number of ASCII digits in array1. array2 will contain the PRIMENET system name. alen2 will contain the number of ASCII characters in array2.

15

IPCF Programming Examples

INTRODUCTION

This chapter contains a general description of programming with IPCF subroutines, and examples, as follows.

- How to establish a virtual circuit
- The general layout of an IPCF application
- Two programming examples

ESTABLISHING A VIRTUAL CIRCUIT

With IPCF subroutines, you can develop applications that consist of modules, with each module running as a separate application on any other or the same PRIMENET system. The different modules use the IPCF subroutines to establish a connection between them. This connection is called a virtual circuit. The major steps in using IPCF subroutines to establish a virtual circuit are listed below.

Caller	Call Receiver
Issue a CALL REQUEST	Receive as INCOMING CALL; respond with CALL ACCEPT
Get CALL CONFIRMATION	
Send and receive data	Send and receive data
Send CLEAR REQUEST	Receive CLEAR INDICATION; send back CLEAR CONFIRMATION
Receive CLEAR CONFIRMATION	

Either of the two applications can initiate a CLEAR REQUEST. Also, the X.25 packet formats of CALL ACCEPT and CALL CONFIRMATION and CLEAR REQUEST and CLEAR INDICATION are identical.

GENERAL LAYOUT OF APPLICATIONS USING IPCF ROUTINES

The IPCF subroutines enable an application to send or receive messages through PRIMENET on an established a virtual circuit. The calling application initiates a call request. The application that is being called assigns a PRIMENET port and waits for incoming calls. Once the virtual circuit is established, either application can transmit data messages. The following example shows the major steps between two IPCF applications attempting to exchange data.

The Calling Application

- Check virtual circuit status array to see if connection is completed.
- Transmit data: X$TRAN.
- Receive data: X$RCV.
- Clear the circuit: XCLR/XFCLR.

The Called Application

- Assign the proper PRIMENET port: X$ASGN.
- Wait in an event-driven loop with X$WAIT. On network events, find data on incoming call with X$ACPT, X$FACP, or XLACPT.
- Accept the connection with X$ACPT, X$FACP, or XLACPT.

- Transmit data: X$TRAN.
- Receive data: X$RCV.
- Clear the circuit: XCLR/XFCLR.

Note

Calls to X$TRAN and X$RCV are made when needed (and in the sequence dictated by the application). An application expecting a message should offer appropriate receive buffers by calling X$RCV.

The calling or called application can terminate the connection by clearing the virtual circuit. It is recommended that the application to receive a message last issue the clear request.

IPCF EXAMPLES

This section contains examples of:

- A simple PRIMENET file-transmission system
- A query and update service on a database using fast-select calls

The purpose of these sample programs is to illustrate typical code for basic IPCF data transfer. The first example contains only the basic code path needed for error-free function. In contrast, the second example has full error handling, following the guidelines set forth in Chapter 16. Also, the general design rules discussed in Chapter 16 are used in the second example. Each example is preceded by a brief description.

PRIMENET FILE-TRANSMISSION SYSTEM

This example consists of two programs, NETSND and NETRCV. Together, they form a rudimentary PRIMENET file-transmission system. They use a common subroutine WAITIL to check for completed network requests. The code of WAITIL follows NETRCV. Notice that the receiver program NETRCV uses double receive buffers to offload PRIMENET.

For simplicity, filenames of only eight characters or less are allowed; access to other directories is not provided for. This example's error handling consists only of STOP statements, which would not be sufficient for a real-life application.

The Transmitting Side

The following program shows an example of the send side of a network copy program.

```
C NETSND.FTN - A SIMPLE, FAST NETWORK FILE COPY PROGRAM
C
C This is the transmitting side of the program; see also NETRCV.
C
$INSERT SYSCOM>X$KEYS.INS.FTN
$INSERT SYSCOM>ERRD.INS.FTN
$INSERT SYSCOM>KEYS.INS.FTN
C
      INTEGER*2 J,FUNIT,CODE,NWR,LEVEL,RSTATE(3),XSTATE,VCID,
     *  VCSTAT(2),FILNAM(4),SYSTEM(3),BUF(1024)
C
C
C The basic idea is to:
C     * Open the requested file in the current ufd.
C     * Establish a circuit with a server on another system.
C     * Send over the filename to the server so it can open
C       a file of the same name for writing.
C     * Send over 1K blocks of the file until we read to the
C       end of the file.
C     * Signal EOF to the server by sending a different
C       user data level.
C     * The server will acknowledge our end of file signal by
C       sending us the code from its close of the target file.
C
C  STOPS are used to signal error conditions:
C        :20 error in circuit establishment
C        :24 error in transmit of filename
C        :25 bad state from transmit of data
C        :30 bad state in acknowledgement receipt
C        :32 bad level in acknowledgement receipt
C        :34 bad length in acknowledgement receipt
C        :50 bad status on clear of virtual circuit
C
C
      FUNIT=1
      WRITE(1,11)
11    FORMAT(' INPUT FILE NAME, 8 CHARS OR LESS')
      READ(1,12) FILNAM              /* Ask for a file to open
12    FORMAT(4A2)
      WRITE(1,13)
13    FORMAT(' INPUT REMOTE SYSTEM NAME')
      READ(1,14) SYSTEM
14    FORMAT(3A2)
C
C  Clear everything
C
      CALL X$CLRA
C
C  Open the file
```

```
C
      CALL SRCH$$(K$READ,FILNAM,8,FUNIT,J,CODE)
C
C  and make sure the file was found.
C
      CALL ERRPR$(K$SRTN,CODE,'ON OPEN',7,0,0)
C
C  Now we set up the virtual circuit.
C  We assume that the receiving half of this program
C  is running on the
C  remote machine and that it has assigned PRIMENET port 10.
C  Connect to remote node. Await non-XS$IP status.
C
      CALL X$CONN(VCID,10,SYSTEM,6,VCSTAT)
      CALL WAITIL(VCSTAT)
      IF (VCSTAT(1).NE.XS$CMP) STOP :20
C
C
C  Send over the filename. Level 1 for control info, level 0 for
C  file data.
C
      CALL X$TRAN(VCID,XT$LV1,FILNAM,8,XSTATE)
C
      CALL WAITIL(XSTATE)
      IF (XSTATE.NE.XS$CMP) STOP :24 /* Could not xmit filename.
C
C
C  Now just keep sending till EOF
C
      LEVEL=XT$LV0                    /* Use data level 0 'til EOF.
C
30    CALL PRWF$$(K$READ,FUNIT,LOC(BUF),1024,000000,NWR,CODE)
C
      IF (CODE.EQ.0) GOTO 35          /* Read OK.
      IF (CODE.NE.E$EOF)              /* If not EOF, it's an error.
     *   CALL ERRPR$(K$SRTN,CODE,'READ',4,0,0)
C
      LEVEL=XT$LV1                    /* Switch LEVEL for EOF.
35    CALL X$TRAN(VCID,LEVEL,BUF,NWR*2,XSTATE)
      CALL WAITIL(XSTATE)
      J=XSTATE
      IF (J.NE.XS$CMP) STOP :25
      IF (LEVEL.EQ.0) GOTO 30         /* Keep sending 'til EOF.
C
C
C  Now wait for acknowledgement from
C  the remote node in the form of a
C  1 word (2 bytes) standard file system error code.
C
      CALL X$RCV(VCID,CODE,2,RSTATE)
      CALL WAITIL(RSTATE)
      IF (RSTATE(1).NE.XS$CMP) STOP :30 /* Bad status
      IF (RSTATE(2).NE.XT$LV1) STOP :32 /* Bad data level for
C                                              control msg
```

```
      IF (RSTATE(3).NE.2) STOP :34      /* Bad length
C
C  Always RETURN to clear the virtual circuit.
C
      CALL ERRPR$(K$IRTN,CODE,'REMOTE CLOSE',13,0,0)
      CALL SRCH$$(K$CLOS,0,0,FUNIT,J,CODE)
      CALL X$CLR(VCID,0,J)              /* Clear virtual circuit.
      IF (J.NE.XS$CMP) STOP :50         /* Check status.
      CALL EXIT
      END
```

The Receiving Side

The following program shows an example of the receiving end of a network copy program.

```
C  NETRCV.FTN - A SIMPLE, FAST NETWORK FILE COPY UTILITY
C
C This is the passive receiving part of a pair of programs.
C The active part, NETSND, runs on another system in this
C network. See NETSND for more information.
C
C
C  The basic idea is to:
C     *  Clear all ports.
C     *  Assign port 10.
C     *  Wait for connection requests.
C     *  Accept the call.
C     *  Get an 8-byte message with the filename to be opened.
C     *  Open the file for writing.
C     *  Receive data messages and write them to the file,
C        while received
C        data messages are not 'USER LEVEL 1'.
C     *  Write a 'LEVEL 1' message to the file.
C     *  Close the file.
C     *  Send an acknowledgement to the sender.
C     *  Wait for the sender to clear the circuit.
C     *  Wait for the next connection request.
C
C STOPS are used to signal error conditions.
C   :10 error on assign of port
C   :14 error in X$GCON call
C   :20 error in X$ACPT call
C   :30 bad receive of filename
C   :32 bad level on receive of filename
C   :34 bad length on receive of filename
C   :40 error while receiving data for the file
C   :50 error transmitting status to sender
C
C NETRCV is designed to be run as a phantom.  The phantom command
C file would have the following format.
C
```

```
C          COMO output file
C          A ufd                    This is the directory files will
C                                   be copied to.
C          EXECUTE NETRCV_runfile
C          LO                       Make sure we log out on error.
C          CO TTY
C
C
C
$INSERT SYSCOM>X$KEYS.INS.FTN
$INSERT SYSCOM>KEYS.INS.FTN
$INSERT SYSCOM>ERRD.INS.FTN
C
      INTEGER BUF(1024,2),CODE,FUNIT,J,Q,RSTATE(3,2),VCSTAT(2),
     +    STAT(2),VCID,I,L
C
C
C  It is a very good programming practice to always have a RECEIVE
C  pending, to relieve the operating system of buffering problems.
C  The buffer and receive status vector will be able to handle two
C  messages at once.
C
      CALL X$CLRA                      /* Clear-up, in case port 10
C                                         previously in use.
      CALL X$ASGN(10,0,J)              /* Assign port 10 forever.
      IF (J.NE.XS$CMP) STOP :10        /* Bad assign.
10    CALL X$WAIT(0)                   /* Wait for connection request.
      CALL X$GCON(VCID,J,STAT)
      L=STAT(1)                        /* Temporary copy.
      IF (L.EQ.XS$NOP) GOTO 10         /* Not really a connect yet.
      IF (L.NE.XS$CMP) STOP :14        /* Bad call to X$GCON.
C
C
C  Here, if we have gotten a connect request.
C
      CALL X$ACPT(VCID,VCSTAT)         /* Accept the connection.
      CALL WAITIL(VCSTAT)              /* Wait 'til not XS$IP.
      L=VCSTAT(1)
      IF (L.NE.XS$IDL .AND. L.NE.XS$CMP)
     *  STOP :20                       /* Some error on ACCEPT.
C
C
C  Now get the file name and open the file.
C
      CALL X$RCV(VCID,BUF(1,1),8,RSTATE(1,1))
      CALL WAITIL(RSTATE(1,1))              /* Wait for not XS$IP.
      IF (RSTATE(1,1).NE.XS$CMP) STOP :30 /* Bad receive.
      IF (RSTATE(2,1).NE.1) STOP :32      /* Wrong level.
      IF (RSTATE(3,1).NE.8) STOP :34      /* Bad length.
C
      FUNIT=3
C     CALL SRCH$$(K$WRIT,BUF(1,1),8,FUNIT,J,CODE) /* Open new file.
      CALL ERRPR$(K$SRTN,CODE,'OPEN',4,0,0)
C
```

```
C
C  Virtual circuit and file are both ready.
C  Let's get into the main copy loop.
C  This loop uses double buffering of receives.
C
C  Start first receive.
C
      CALL X$RCV(VCID,BUF(1,1),2048,RSTATE(1,1))
C
C  Start second receive.
C
      CALL X$RCV(VCID,BUF(1,2),2048,RSTATE(1,2))
      I=2                              /* Init double buffer pointer.
C
20    I=I+1                            /* Indexes the buffer number.
      IF (I.EQ.3) I=1                  /* Flip flops 1,2,1,2,1,2....
      CALL WAITIL(RSTATE(1,I))
      J=RSTATE(1,I)
      IF (J.NE.XS$CMP) STOP :40        /* Error on RCV.
      L=RSTATE(3,I)/2                  /* Convert bytes to words.
      CALL PRWF$$(K$WRIT,FUNIT,LOC(BUF(1,I)),L,000000,J,CODE)
      CALL ERRPR$(K$SRTN,CODE,'WRITE',5,0,0)
      IF (RSTATE(2,I).EQ.XT$LV1)  GOTO 30       /* Level 1 <=> EOF.
      CALL X$RCV(VCID,BUF(1,I),2048,RSTATE(1,I)) /* Issue receive.
      GOTO 20
C
C
30    CALL SRCH$$(K$CLOS,0,0,FUNIT,J,CODE)  /* Close the file.
      CALL X$TRAN(VCID,XT$LV1,CODE,2,J) /* Return close code
C                                                  to sender.
      CALL WAITIL(J)                /* Wait for transmit complete.
      IF (J.NE.XS$CMP) STOP :50     /* Error on transmit.
50    CALL X$WAIT(0)                /* Wait for CLEAR from sender.
      J=VCSTAT(1)                   /* Copy over circuit status.
      IF (J.EQ.XS$CLR) GOTO 10      /* OK to accept next call.
      IF (J.EQ.XS$CMP) GOTO 50      /* Wait for CLEAR,
      STOP :54                      /* else some circuit error.
      END
```

Routine to Wait for Next Network Event

The following subroutine is part of the fast select networking program.

```
C WAITIL.FTN
C
C This subroutine returns when its argument
C (an asynchronously updated
C network status word) is anything other than 'XS$IP'.
C
      SUBROUTINE WAITIL(SIWORD)
C
$INSERT SYSCOM>X$KEYS.INS.FTN
```

```
C
      INTEGER STWORD
      IF (STWORD.NE.XS$IP) RETURN   /* Nothing to wait for? Don't,
C                                      unless process gets hung on
C                                      the semaphore.
10    CALL X$WAIT(0)
      IF (STWORD.EQ.XS$IP) GOTO 10
      RETURN
      END
```

FAST SELECT CALLS

The purpose of this example is to illustrate the use of fast select calls and the corresponding short form IPCF subroutines. In addition, a call passoff to a second server type illustrates the use of XLGVVC. The protocol at the user level is designed to minimize network overhead and connection time. The intention of this scenario is to offer a set of example programs that provide a query and update service on a database, with the users spread over a network but the database centralized to one node. The majority of the transactions are supposed to be status questions, which have brief answers. However, there are also updates and queries that have long answers.

The implementation contains

- A user program
- A query server (single-threaded, running in multiple invocations)
- One update server (multi-task design)

The user program handles screen layouts and data compression/expansion. The user program connects to a query server to obtain the answer or perform the update. The transaction input data always fits into a record of 80 bytes. The brief answers occupy a record of size 100 bytes. Other (long) answers are of various lengths. Updates also require answers of varying lengths. These long answers are at most 2000 bytes long. They are transferred as single IPCF messages, and the "receive complete" status indicates the end of transfer.

The majority of exchanged data records will be either 80 or 100 bytes, both of which fit into the user data fields of fast select calls. Thus questions with brief answers are handled as fast select calls, given a fast clear by the query server. Long answers and updates are also initiated as fast select calls, but these are accepted, and the answer message from the server is cleared by the user on receipt.

Unknown to the users, the system designer has decided to run all updates by a separate single update server, having the multiple 'query' servers to detect update calls and pass these off to the update server.

Capacity and Service Busy

Several query servers run to handle the stream of queries. The number of servers is expected to be so large that the probability of no available server is acceptably low. Consequently, there is no special server to indicate "service busy". The application will deduce from the clearing diagnostic CD$PNA (port not assigned) that all query servers are currently busy.

There will be only one update server. However, this server can have multiple active requests going. Again, the number of query servers is expected to be sufficiently large; and, if it is exceeded, the update server will clear the new call with an application-specific diagnostic (CD$NVC). If the update server is not running, any query server that fails to pass an update off will clear the call with the same diagnostic.

Timing Aspects

The protocol attempts to keep turnaround times small. In fact, the majority of transactions, queries with brief answers, are handled with only two information-carrying packets, the call request and the clear request. Notice that long answers are submitted to PRIMENET as single messages, and that the final user side acknowledgement of a long answer is combined with the clear request, generated by the user program.

The query server extracts the answer to a question by a subroutine call, before either clearing or accepting the virtual circuit. This means that this database routine must return within the timeframe set by PRIMENET for a user process to act on a call request. Otherwise, PRIMENET does a 'safety' clear, and the user will get no answer. Similarly, the update server is busy when calling the update routine. This means that passed-off call requests will temporarily hang, and the same maximum timeframe for delays exists here.

If the update times tend to be considerable, you might consider splitting the update server into two processes, one dealing with PRIMENET and the other with the data base. The network process would simply queue updates for its mate, and sort responses on appropriate virtual circuits for transmission. The link between these two update mates could either be a virtual circuit, the file system, or shared memory with write access. The program structure of the network handler would remain much the same, with additional code in the "per active virtual circuit" loop, to find completed answers and transmit them.

Virtual Circuit Timeout Handling

Notice also the difference in handling aging virtual circuits. The query server is a 'single-path' program, and the final timeout before safety clearing is simply implemented as a long wait on the network semaphore, combined with a safety call to X$CLRA before restarting.

In contrast, the update server must run frequently to ensure that every virtual circuit moves along. If awakened, it may well be on behalf of another virtual circuit, so it is difficult to implement long timeouts by using the network semaphore. The solution chosen is to run a watchdog timer for each circuit, and to force the clear when this expires. Furthermore, this server must not call X$CLRA, as this would obviously kill all active circuits.

The Code

The routines for handling the user terminal and the data base are omitted. The names of the tasks are noted. The programs are coded in F77. Extensive use of INTS has been made, to remind the reader that F77's default integer mode of INTEGER*4 causes a risk for wrong argument values when called routines expect INTEGER*2 arguments, as the IPCF routines do.

Common Insert Files:

```
C FSX_DATA.INS.F77, Common declaration of action keys and
C query message structures for a fast select program example.
C
      NOLIST
C
C PRIMENET node and port data. Values to be
C tailored by installation.
C
      INTEGER*2 query_port, update_port
      PARAMETER (query_port = $$, update_port = &&)
C
      CHARACTER*6 server_node /'XXXXXX'/
C
C Action keys, contained in request message and initial response.
C
      INTEGER*2 query, update, brief_response, long_response,
     +      update_started, exit
      PARAMETER (query = 1, update = 2, brief_response = 3,
     +      long_response = 4, update_started = 5, exit = 6)
C
C Clearing diagnostics, used for various return status messages.
C
      INTEGER*2 CD$SHR, CD$LNG, CD$EOU, CD$IMO, CD$RST, CD$NVC
      PARAMETER (CD$SHR = 1,     /* Fast clear, proper short reply
```

```
     +      CD$LNG = 2,          /* Clear, ack of long reply
     +      CD$ECU = 3,          /* Clear, ack of update response
     +      CD$IMO = 4,          /* Clear by impatient server
     +      CD$RST = 5,          /* Clear after reset
     +      CD$NVC = 6)          /* Update server has no VC
C                                 /* (or is not running)
C
C User to server request message (80 bytes, the first two to
C hold the action_key).
C
      INTEGER*2 msg_size, data_size
      PARAMETER (msg_size = 80, data_size = msg_size-2)
C
      INTEGER*2 message(msg_size/2), action_key,
     +      data_string(data_size/2)
C
      EQUIVALENCE (action_key, message(1)),
     +      (data_string(1), message(2))
C
C Server to user response:
C [brief_response]:
C Brief response is 100 bytes, first two to hold response key.
C [long_response]:
C Long response is up to 2000 bytes, -"-
C     and the first 100 come in the fast accept, the rest by X$RCV.
C [update_started]:
C Two bytes only, the rest by later X$RCV.
C
      INTEGER*2 resp_size, answer_size, total_size, remndr_size
      PARAMETER (resp_size = 100, answer_size = resp_size-2,
     +    total_size = 2000, remndr_size = total_size - resp_size)
C
      INTEGER*2 response(total_size/2), response_key,
     +    answer(answer_size/2), remainder(remndr_size/2)
C
      EQUIVALENCE (response_key, response(1)),
     +    (answer(1), response(2)),
     +    (remainder(1), response(resp_size/2+1))

C RETURNED is the array to catch the retrieved
C user data field (rudat).
C By equivalencing, the interesting response
C is extracted from behind the four protocol id bytes.
C
      INTEGER*2 return_size
      PARAMETER (return_size = resp_size + 4)
C
      INTEGER*2  returned(return_size/2)
      EQUIVALENCE (response(1), returned(3))
C
      LIST
```

```
C MULTI_VC.INS.F77, multiple VC database for update server
C
      NOLIST
C
      INTEGER*2 pool_size
      PARAMETER (pool_size = 10)
C
      COMMON /ADMNVC/ next_free, total_used
      COMMON /MANYVC/ in_use(pool_size),
     +      vc_id(pool_size),
     +      vc_status(2, pool_size),
     +      xmit_status(pool_size),
     +      zero_time(pool_size)
C
      LOGICAL*2 in_use
      INTEGER*2 next_free, total_used, vc_id, vc_status,
     +      xmit_status, zero_time
C
      LOGICAL*2 GETVC
      INTEGER*2 AGE
      EXTERNAL INITVC, FREEVC, ORGSTAMP
C
      LIST
```

```
C UPDATE_DATA.INS.F77, Per virtual circuit update message buffers
C Requires the previous insert of (multi_vc fsx_data).INS.F77.
C
      NOLIST
C
      COMMON /UPDIN/ in_msg(msg_size, pool_size)
      INTEGER*2 in_msg
C
      COMMON /UPDOUT/ out_msg(total_size, pool_size)
      INTEGER*2 out_msg
C
      LIST
```

User Program:

```
C FS_USER.F77, Sample program for fast-select PRIMENET connections.
C
C This is the interactive user program, to be run interactively
C at need. Any user action initiates a fast select call to one of
C the query_servers, which will either respond or pass over to the
C one and only update_server.
C
C Short replies arrive back with a fast clear. Updates and long
C answers are both accepted and yield further data transfer,
C in the form of long messages.
C On receiving the message, the user acknowledges
```

```
C by a normal clear with the appropriate diagnostic.
C
      PROGRAM main
C
$INSERT syscom>x$keys.ins.ftn
$INSERT *>fsx_data.ins.f77
C
      INTEGER*2 user_vc, vc_status(2), rcv_stat(3), statword,
     +      timeout, actual_ret_size, temp, clr_cause, clr_diag
C
      INTEGER*2 X$WAIT
C
      INTRINSIC INTS, INTL, LT, RT
      EXTERNAL X$WAIT, X$FCON, X$RCV, X$CLR, X$CLRA, SLEEP$, TNOU
      EXTERNAL getinput, dispdata, dispansw, noserver, nepr
C
C
C Keep executing this until the user says 'exit'.
C
1     CONTINUE
C
C Get the user input - this will include screen formatting.
C
      CALL getinput(message)
C
      IF (action_key .EQ. exit) RETURN
C
C Whether the user asks for a query or update, we call the
C query server, which will sort things out by itself.
C
      CALL X$FCON(XK$NAM+XK$ANY, XK$ACC, user_vc,
     +      query_port, server_node, INTS(6), message, msg_size,
     +      vc_status,
     +      returned, return_size, actual_ret_size)
C
C Status test:
C XS$CMP and XS$CLR imply completed connection;
C XS$IP: wait for PRIMENET to complete the connection,
C then get a new copy of the VC status;
C All others: crash.  Clear everything and return to command level.
C
100   temp = vc_status(1)              /* Get local copy.
C
      IF (temp .EQ. XS$IP) THEN
         timeout = X$WAIT(INTS(100))  /* Arbitrary 10 seconds.
         GO TO 100
      ELSE IF (temp .EQ. XS$CLR) THEN
         GO TO 150
      ELSE IF (temp .EQ. XS$CMP) THEN
         GO TO 200
      ELSE
         CALL nepr(vc_status, INTS(2))
         GO TO 9000
      ENDIF
```

```
C
C ---   ---   ---
C Some sort of connection success has occurred (XS$CLR or XS$CMP).
C If the connect was completed, this means that the server accepted
C the call, to do further data transfer.
C If the connect was cleared, this could either be by a failure,
C or by a fast select clear.
C In the latter case, there are data to display.
C
C ---
C Cleared connection:
C IF the cause and diagnostic are correct, display results and
C restart, else restart.
C
150   clr_cause = LT(vc_status(2), 8)
      clr_diag = RT(vc_status(2), 8)
C
    IF (clr_cause .EQ. CC$CLR .AND. clr_diag .EQ. CD$SHR) THEN
         CALL dispansw(response)
         GO TO 1
    ELSE IF (clr_cause .EQ. CC$CLR .AND. clr_diag .EQ. CD$PNA) THEN
         CALL noserver
         GO TO 1
    ELSE
         CALL nepr(vc_status, INTS(2))
         GO TO 400
    ENDIF
C
C ---
C Accepted connection:
C Issue receive call for returned data, then display the result.
C The invented protocol controls the choice of array to supply to
C X$RCV.
C EITHER it is a long answer, in which case the beginning is
C already here, OR it is just the key "update_started" and the
C whole answer, except the action key, will come later.
C
200   IF (actual_ret_size .LT. INTS(6)) GO TO 400  /* Key missing!
C
210   IF (response_key .EQ. update_started) THEN
         CALL X$RCV(user_vc, answer, answer_size + remndr_size,
     +      rcv_stat)
      ELSE
         CALL X$RCV(user_vc, remainder, remndr_size, rcv_stat)
      ENDIF
C
C Status test:
C XS$IP means still coming in, wait a bit( on network semaphore).
C XS$CMP is OK - all done, now ack by a clear and then restart.
C XS$RST means Reset occurred: clear the circuit.
C XS$CLR is not anticipated - restart.
C XS$MEM means attempt failed, retry the receive shortly.
C All others, fatal crash!
C
```

```
220   temp = rcv_stat(1)
      IF (temp .EQ. XS$IP) THEN
         timeout = X$WAIT INTS(20)          /* Arbitrary 2 sec.
         GO TO 220
      ELSE IF (temp .EQ. XS$CMP) THEN
         GO TO 230
      ELSE IF (temp .EQ. XS$CLR) THEN
         CALL nepr(vc_status, INTS(2))
         GO TO 400
      ELSE IF (temp .EQ. XS$MEM) THEN
         CALL SLEEP$(INTL(1000))            /* Wait a sec...
         GO TO 210
      ELSE IF (temp .EQ. XS$RST) THEN       /* Reset occurred, clear.
         CALL X$CLR(user_vc, CD$RST, statword)
         IF (statword .NE. XS$CMP) THEN
            CALL nepr(statword, INTS(1))
            GO TO 9000
         ENDIF
         GO TO 240                          /* Await confirmation.
      ELSE
         CALL nepr(rcv_stat, INTS(3))
         CALL nepr(vc_status, INTS(2))
         GO TO 9000
      ENDIF
C
C Receive complete; now send acknowledging clear, and display data.
C Tell the display routine the entire response length.
C
230   IF (response_key .EQ. update_started) THEN
         CALL X$CLR(user_vc, CD$EOU, statword)
         CALL dispdata(answer, resp_size + rcv_stat(3))
      ELSE
         CALL X$CLR(user_vc, CD$LNG, statword)
         CALL dispdata(answer, rcv_stat(3))
      ENDIF
C
C Verify the correct status for the clear request.
C The only reasonable return code here is XS$CMP.
C
      IF (statword .EQ. XS$CMP) THEN
         GO TO 240                          /* Await confirmation.
      ELSE
         CALL nepr(statword, INTS(1))
         GO TO 9000                         /* Fatal error.
      ENDIF
C
C Tidy up for restart; await confirmation of requested clear.
C If not arrived in 30 seconds (will the user stand more?),
C forget the circuit and restart.
C
240   IF (vc_status(1) .NE. XS$CLR) THEN
         PRINT 2000
         timeout = X$WAIT(INTS(300))
      ENDIF
```

```
2000  FORMAT ('Disconnecting...')
C
      IF (vc_status(1) .NE. XS$CLR)
     +      PRINT 2010
2010  FORMAT ('Clear request unconfirmed - restarting')
      CALL X$CLRA
      GO TO 1                  /* Restart
C
C
C - - - - - - - -
C Protocol problems or network transmission problems.
C
400   CALL TNOU('Transfer failure', INTS(16))
      CALL X$CLRA
      GO TO 1                  /* Restart
C
C
C = = = = = = = =
C Fatal errors: crash exit to command level
C after global network cleanup.
C
9000  CALL TNOU('Network failure', INTS(15))
      CALL X$CLRA
      RETURN
C
      END
```

Query Server:

```
C QUERY_SERVER.F77, sample server program for
C fast-select PRIMENET connections.
C
C The query_server program is expected to run as several
C parallel processes. Therefore, it assigns its port for
C one call only, and always reassigns on completed service.
C Enough servers are expected to be running to ensure
C 100 percent availability. If, however, the application
C runs out of servers, the PRIMENET clearing with CD$PNA
C (port not assigned) will indicate no servers.
C
C The first action is to detect updates, and to pass these
C to the update server. If the passoff fails, the circuit
C is cleared with a special diagnostic.
C
C Depending on the query, it will either send a short answer
C by a fast select clear or accept the call with part of the
C answer, and then send the rest separately. To ensure that
C the full answer gets through, the user clears in this case,
C thereby acknowledging. The server has a safety timeout, as well.
C
      PROGRAM main
C
```

```
$INSERT syscom>x$keys.ins.ftn
$INSERT *>fsx_data.ins.f77
C
      INTEGER*2 statword, status(2), vc_status(2), xmt_status,
     +      answer_key, server_vc, dummy_port, rn_len, msg_bytes,
     +      rem_length, clr_cause, clr_diag, timeout,
     +      not_used, must_be_0, temp, junk
      CHARACTER*6 remote_node
      LOGICAL*2 long_flag
C
      PARAMETER (must_be_0 = 0)
C
      INTRINSIC INTL, INTS, LT, RT
      INTEGER*2 X$WAIT
      EXTERNAL X$ASGN, X$WAIT, X$FGCN, X$FCLR, X$FACP, XLGVVC,
     +      X$TRAN, X$CLR, X$CLRA, SLEEP$
      EXTERNAL dbanswer, nepr
C
C
C Restart point - assign the query server port to take ONE call.
C
1     CALL X$ASGN(query_port, INTS(1), statword)
C
C Error test: XS$CMP or XS$QUE are satisfactory.  For all others,
C fatal crash.
C
      IF (statword .NE. XS$CMP .AND. statword .NE. XS$QUE) THEN
         CALL nepr(statword, INTS(1))
         GO TO 9000
      ENDIF
C
C Wait for somebody to call. When awakened from the
C network semaphore, find out about the call.
C (For safety, make routine wake-up once every minute.)
C
10    timeout = X$WAIT(INTS(600))
      CALL X$FGCN(XK$NAM, answer_key, server_vc, dummy_port,
     +      remote_node, INTS(6), rn_len,
     +      message, msg_size, msg_bytes,
     +      status)
C
C Status test: XS$NOP means spurious wake up -> wait more.
C              XS$CMP means call to handle.
C For all others, fatal crash.
C
      temp = status(1)
      IF (temp .EQ. XS$NOP) THEN
         GO TO 10
      ELSE IF (temp .EQ. XS$CMP) THEN
         GO TO 100
      ELSE
         CALL nepr(status, INTS(2))
         GO TO 9000
      ENDIF
```

```
C
C ---
C Handle the call. Sort out updates, pass them off
C to the query server. Otherwise send the message to
C the data base, retrieve the answer, and analyze its length.
C Long answers force call accepted, with later transmission
C of the actual answer, short ones are 'fast_cleared'
C carrying the answer.
C
100   IF (message(1) .EQ. update) GO TO 300
C
      CALL dbanswer(message, response, long_flag, rem_length)
      IF (long_flag) GO TO 200
C
C ---
C Short answer <-> fast clear
C Here we use the "returned" aggregate array,
C not to place data overlapping PRID field.
C Also, the diagnostic code should be CD$SHR,
C for the user's validity test.
C
110 CALL X$FCLR(server_vc, CD$SHR, returned, return_size, statword)
C
C Status test:
C XS$CMP is OK - all done, restart
C NOTE: There is no vc_status array set up yet, so we cannot
C check for clear confirmation!
C XS$MEM means attempt failed, retry soon.
C For all others, fatal crash.
C
      temp = statword
      IF (temp .EQ. XS$CMP) THEN
         GO TO 8000
      ELSE IF (temp .EQ. XS$MEM) THEN
         CALL SLEEP$(INTL(1000))                    /* Wait a sec...
         GO TO 110
      ELSE
         CALL nepr(statword, INTS(1))
         GO TO 9000
      ENDIF
C
C ---
C Long answer:
C Accept the call, and then transmit the rest of the long answer.
C
200   CALL X$FACP(server_vc, response, resp_size, vc_status)
C
C Status test:
C XS$IDL/XS$CMP is OK - ready to transmit data.
C XS$MEM means attempt failed, retry shortly.
C XS$CLR is not expected, but let the server live
C to do further work.
C For all others, fatal crash.
C
```

```
      temp = vc_status(1)                          /* Get local copy.
      IF (temp .EQ. XS$IDL .OR. temp .EQ. XS$CMP) THEN
         GO TO 210
      ELSE IF (temp .EQ. XS$MEM) THEN
         CALL SLEEP$(INTL(1000))                   /* Wait a sec...
         GO TO 200
      ELSE IF (temp .EQ. XS$CLR) THEN
         CALL nepr(vc_status, INTS(2))
         GO TO 8000                                /* To restart.
      ELSE
         CALL nepr(vc_status, INTS(2))
         GO TO 9000                                /* Fatal, death.
      ENDIF
C
210   CALL X$TRAN(server_vc, XT$LV0,
     +      remainder, rem_length - resp_size, xmt_status)
C
C Status test:
C XS$IP means still pushing it off, wait on network semaphore.
C XS$CMP is OK - all done, now await confirming clear and restart.
C XS$CLR - check if it was the "ack", then restart the server.
C XS$RST - reset occurred, clear the circuit.
C XS$MEM means attempt failed, retry shortly.
C For all others, fatal crash.
C
220   temp = xmt_status
      IF (temp .EQ. XS$IP) THEN
         timeout = X$WAIT(20)                /* Arbitrary 2 seconds.
         GO TO 220
      ELSE IF (temp .EQ. XS$CMP) THEN
         GO TO 230
      ELSE IF (temp .EQ. XS$CLR) THEN
         GO TO 240
      ELSE IF (temp .EQ. XS$MEM) THEN
         CALL SLEEP$(INTL(1000))             /* Wait a sec...
         GO TO 210
      ELSE IF (temp .EQ. XS$RST) THEN        /* On reset, clear.
         CALL X$CLR(server_vc, CD$RST, statword)
         IF (statword .NE. XS$CMP) THEN
            CALL nepr(statword, INTS(1))
            GO TO 9000
         ENDIF
         GO TO 7000                          /* Await confirmation.
      ELSE
         CALL nepr(vc_status, INTS(2))
         GO TO 9000                          /* Fatal, death.
      ENDIF
C
C Transmit complete. Now wait for the acknowledging clear
C using the CD$LNG diagnostic. If too long a time passes,
C then clear from this end, and restart.
C (Ensure the network semaphore is drained before starting timeout
C waiting of 2 minutes.)
C
```

```
230   timeout = X$WAIT(INTS(1))
      IF (vc_status(1) .EQ. XS$CLR) THEN
         GO TO 240                        /* Verify the diagnostic.
C
      ELSE
         timeout = X$WAIT(INTS(1200))     /* Give the user 2 minutes.
         IF (vc_status(1) .EQ. XS$CLR) THEN
            GO TO 240                     /* Verify the diagnostic.
         ELSE
            PRINT 2300
            CALL X$CLR(server_vc, CD$TMO, statword)
C
C The only reasonable return code here is XS$CMP.
C
            IF (statword .EQ. XS$CMP) THEN
               GO TO 7000                 /* Await confirmation.
            ELSE
               CALL nepr(statword, INTS(1))
               GO TO 9000                 /* Fatal error.
            ENDIF
         ENDIF
      ENDIF
2300  FORMAT ('Forced server clear...')
C
C Verify cleared by user with correct diagnostic.
C If it is not normal, print the message,
C then restart in any case.
C
240   clr_cause = LT(vc_status(2), 8)
      clr_diag = RT(vc_status(2), 8)
C
      IF (clr_cause .NE. CC$CLR .OR. clr_diag .NE. CD$LNG)
     +      CALL nepr(vc_status, INTS(2))
      GO TO 8000
C
C ---   ---   ---
C Pass over an update.  This is done before call acceptance, and
C by port number. It requires XLGVVC, and since the original call
C request packet is still around, we are not allowed to try
C providing a new one.
C
300   CALL XLGVVC(XK$PRT, server_vc, not_used, must_be_0, must_be_0,
     +      update_port,
     +      junk, INTS(0), junk, INTS(0),       /* Unsupplied
     +      junk, INTS(0), junk, INTS(0),       /* packet
     +      junk, INTS(0),                      /* fields.
     +      statword)
C
C Status test:
C XS$CMP is OK - all done, restart from the beginning.
C XS$UNK implies that an update server is not running,
C clears with CD$NVC.
C XS$MEM means attempt failed, retry shortly.
C For all others, fatal crash.
```

```
C
      temp = statword
      IF (temp .EQ. XS$CMP) THEN
         GO TO 8000
      ELSE IF (temp .EQ. XS$MEM) THEN
         CALL SLEEP$(INTL(1000))          /* Wait a sec...
         GO TO 300
C
      ELSE IF (temp .EQ. XS$UNK) THEN
         CALL X$CLR(server_vc, CD$NVC, statword)
C
C The only reasonable return code here is XS$CMP.
C There is no vc_status array set up yet, so we cannot check
C for clear confirmation!
C
         IF (statword .EQ. XS$CMP) THEN
            GO TO 8000                    /* Restart immediately.
         ELSE
            CALL nepr(statword, INTS(1))
            GO TO 9000                    /* Fatal error.
         ENDIF
C
      ELSE
           CALL nepr(statword, INTS(1))
           GO TO 9000                     /* Fatal error.
      ENDIF
C
C
C = = = = =
C Tidy up for restart.  Await confirmation of requested clear.
C If it has not arrived in 2 minutes, forget the circuit and
C restart.  (First wait a very short time,
C to ensure the network semaphore
C gets drained from the previous notifications.)
C
7000  timeout = X$WAIT(INTS(1))
      IF (vc_status(1) .EQ. XS$CLR) THEN
         GO TO 8000                       /* That's it!
C
      ELSE
         timeout = X$WAIT(INTS(1200))
         IF (vc_status(1) .EQ. XS$CLR) THEN
            GO TO 8000
         ELSE
            IF (timeout .NE. 0) PRINT 7010
            CALL nepr(vc_status, INTS(2))
            PRINT 7020
            GO TO 8000
         ENDIF
      ENDIF
C
7010  FORMAT ('Two minutes time-out...')
7020  FORMAT ('Clear request unconfirmed - restarting')
C
```

```
C --
C General restart code.  Ensure we start in fresh environment by
C calling X$CLRA.
C
8000  CALL X$CLRA
      GO TO 1
C
C
C * * * * *
C Fatal error - print message and die...
C
9000  PRINT 9010
9010  FORMAT  ('Fatal error')
      CALL X$CLRA
      RETURN
C
      END
```

Update Server:

```
C UPDATE_SERVER.F77, Example server program for fast-select
C PRIMENET connections.
C
C There is only one update server function, which thus must be
C able to handle multiple virtual circuits.
C
C The life of each individual virtual circuit is:
C X$FGCN -> X$FACP -> do update -> X$TRAN, wait for XS$CMP ->
C -> wait for clear request -> back to free pool...
C In case of the user failing to clear, there must also be a
C timeout mechanism for the virtual circuit to be released.
C
C A mechanism for allocating and releasing sets of status
C arrays per VC is used, making use of the variable 'next_free'.
C Since only a finite pool of VCs can be run, calls
C arriving when the pool is fully used are cleared,
C with the diagnostic CD$NVC.
C
C The main structure of this server is to run a service
C loop, paced by the network semaphore.
C On wake_up it will look for new connections
C and then check on all active virtual circuits.
C
      PROGRAM main
C
$INSERT syscom>x$keys.ins.ftn
$INSERT *>fsx_data.ins.f77
$INSERT *>multi_vc.ins.f77
$INSERT *>update_data.ins.f77
C
      INTEGER*2 statword, status(2),
     +    server_vc, answer_key, dummy_port, rn_len, msg_bytes,
```

```
     +    clr_cause, clr_diag, junk, index, update_length, temp, i
C
      CHARACTER*6 remote_node
C
      INTRINSIC INTS, INTL, LT, RT
      EXTERNAL X$ASGN, X$WAIT, X$FCLR, X$FACP, X$TRAN, X$CLR,
     +      X$CLRA, X$FGCN, SLEEP$
      EXTERNAL do_updat, nepr, getvc, age
C
C Initialize the virtual circuit pool!
C
      CALL initvc
C
C Assign the update server port to take ALL calls.
C
      CALL X$ASGN(update_port, INTS(0), statword)
C
C Error test: XS$CMP is satisfactory, for all others, fatal crash.
C XS$QUE is not legal, since there can be only one update server.
C
      IF (statword .NE. XS$CMP) THEN
         CALL nepr(statword, INTS(1))
         GO TO 9000
      ENDIF
C
C
C START OF MAIN SERVICE LOOP. = = = = = = = = = = = = = = = = = = = =
 =
C Wait on network event, five_second safety time-out. (NOTE: This
C timeout MUST be short, since it controls the speed of servicing
C running virtual circuits.)
C
10    CALL X$WAIT(INTS(50))
C
C ---   ---   ---
C Look for new incoming calls. If VC available, accept or clear it.
C Carry on until calls are exhausted or when VCs are unavailable.
C
50    CALL X$FGCN(XK$NAM, answer_key, server_vc, dummy_port,
     +      remote_node, INTS(6), rn_len,   /* Collect call origin.
     +      message, msg_size, msg_bytes,
     +      status)
C
C Status test.
C XS$CMP means call to handle.
C XS$NOP means spurious wake up or calls exhausted.
C Do the service loop for active VCs.
C For all others, fatal crash.
C
      IF (status(1) .EQ. XS$NOP) THEN
         GO TO 200
      ELSE IF (status(1) .EQ. XS$CMP) THEN
         GO TO 100
      ELSE
```

```
        CALL nepr(status, INTS(2))
        GO TO 9000
      ENDIF
C
C ---
C Handle the call. If VC available, accept it, else clear.
C
100   IF (.NOT. getvc(server_vc, index)) THEN
C
        CALL X$CLR(server_vc, CD$NVC, statword)
C
C The only reasonable return code here is XS$CMP.
C There is no vc_status array set up yet, so we cannot check
C for clear confirmation!
C
        IF (statword .EQ. XS$CMP) THEN
          GO TO 50                        /* Look for further calls.
        ELSE
          CALL nepr(statword, INTS(1))
          GO TO 9000                      /* Fatal, death.
        ENDIF
C
      ELSE
        out_msg(1,index) = update_started
105     CALL X$FACP(server_vc, out_msg(1,index), INTS(2),
     +      vc_status(1, index))
C
C Status test:
C XS$IDL/XS$CMP is OK - all done, restart from the beginning.
C XS$MEM means attempt failed, retry.
C XS$CLR is not expected, but let us carry on.
C For all others, fatal crash.
C
        temp = vc_status(1,index)         /* Get local copy.
C
        IF (temp .EQ. XS$IDL .OR. temp .EQ. XS$CMP) THEN
          GO TO 110
        ELSE IF (temp .EQ. XS$MEM) THEN
          CALL SLEEP$(INTL(1000))         /* Wait a sec...
          GO TO 105
        ELSE IF (temp .EQ. XS$CLR) THEN
C
C Unexpected. Look for more calls. The virtual circuit will be
C released in the "test if cleared" loop further below.
C
          GO TO 50
        ELSE
          CALL nepr(vc_status(1, index), INTS(2))
          GO TO 9000                      /* Fatal, death.
        ENDIF
C
C
C Circuit set up. Get the update done, transmit the update message,
C and start the clock for "time since transmit".
```

```
C
C (Note that the following do_updat call implies that "message" is
C copied to "in_msg(.,index)" BEFORE do_updat returns, so that
C "message" then can be reused.)
C
110      CALL do_updat(message, msg_bytes, index, update_length)
C
         CALL X$TRAN(server_vc, XT$LV0,
     +      out_msg(2,index), update_length, xmit_status(index))
         CALL orgstamp(index)
C
C To keep service speed up, we should NOT wait for XS$CMP here, but
C only sort out fatal errors. 'Expected' statuses will be handled
C later in the general all-VC loop.
C
         temp = xmit_status(index)
         IF (temp .EQ. XS$IP .OR. temp .EQ. XS$CMP .OR.
     +      temp .EQ. XS$CLR .OR. temp .EQ. XS$BVC .OR.
     +      temp .eq. XS$RST) THEN
            GO TO 50                        /* Look for more calls.
         ELSE
            CALL nepr(xmit_status(index), INTS(1))
            GO TO 9000                      /* Fatal, death....
C
         ENDIF                          /* xmit-status.
C
      ENDIF                             /* clear/accept.
C
C
C ---   ---   ---
C Loop for running VCs:
C This loop MUST ensure that all VCs terminate properly and are
C released.
C - If the transmit completes and the user does not clear,
C   the server should clear;
C - if the transmit does not complete, the server should clear;
C - if resets occur, the server should clear.
C In this way the circuit will always have a clear request;
C the timer mechanism of PRIMENET (19.3 onwards) will then
C ensure that the confirming state XS$CLR is reached
C (with appropriate diagnostic), and the VC will be released.
C
200   DO 250 i = 1, pool_size
      IF (in_use(i)) THEN               /* Skip inactive circuits.
C
C Check that the VC is cleared. Print error message
C if there is a wrong cause or diagnostic. Free it in any case.
C
         temp = vc_status(1, i)
         IF (temp .EQ. XS$CLR) THEN
C
            clr_cause = LT(vc_status(2, i), 8)
            clr_diag = RT(vc_status(2, i), 8)
            IF (clr_cause .NE. CC$CLR .OR. clr_diag .NE. CD$EOU)
```

```
     +            CALL nepr(vc_status, INTS(2))
          CALL freevc(i)
C
C Clear a virtual circuit that was reset. The circuit will then be
C released in a subsequent test loop.
C
        ELSE IF (temp .EQ. XS$RST) THEN
           CALL X$CLR(vc_id(i), CD$RST, statword)
           IF (statword .NE. XS$CMP) THEN
              CALL nepr(statword, INTS(1))
              GO TO 9000
           ENDIF
C
C Now look at the transmit status.
C
        ELSE
           junk = xmit_status(i)
C
C We can neglect XS$RST and XS$CLR, since they were already trapped
C in the previous vc_status test.
C
C If this transmit has XS$BVC, the cause is likely to be a clear
C before the transmit was attempted. This should already have
C been detected in the previous vc_status test, but "just in case"
C ensure the circuit becomes free.
C
           IF (junk .EQ. XS$BVC) THEN
              CALL freevc(i)
C
C If the transmit does not complete in decent time, we suspect a
C "hang". Clear the circuit, and to prevent repeats, CHANGE the
C TRANSMIT status to XS$CLR. The time limit is set to 5 minutes
C (arbitrary choice).
C
          ELSE IF (junk .EQ. XS$IP .AND. age(i) .GE. INTS(5)) THEN
              CALL X$CLR(vc_id(i), CD$TMO, statword)
              IF (statword .NE. XS$CMP) THEN
                 CALL nepr(statword, INTS(1))
                 GO TO 9000
              ENDIF
              xmit_status(i) = XS$CLR
C
C Similarly, if the transmit completed but no clear arrives, again
C we clear a suspected "hang". To prevent repeats, CHANGE the
C TRANSMIT status to XS$CLR. The time limit is set to the same
C 5 minutes (arbitrary choice).
C
          ELSE IF (junk .EQ. XS$CMP .AND. age(i) .GE. INTS(5)) THEN
              CALL X$CLR(vc_id(i), CD$TMO, statword)
              IF (statword .NE. XS$CMP) THEN
                 CALL nepr(statword, INTS(1))
                 GO TO 9000
              ENDIF
              xmit_status(i) = XS$CLR
```

```
C
            ENDIF       /* Transmit status testing.
C
          ENDIF         /* VC status testing.
C
        ENDIF           /* VC active.
C
250     CONTINUE
C
C All done. Go back to sleep.
C
        GO TO 10
C
C * * * * *
C Fatal error, death....
C
9000    PRINT 9010
9010    FORMAT  ('Fatal error!')
        CALL X$CLRA
        RETURN
C
        END
```

VC Pool Handling:

```
C HANDLE_VC.F77, subroutines to allocate update server VCs
C
C INITVC - initialize the VC flag and status database

        SUBROUTINE INITVC
C
$INSERT *>MULTI_VC.INS.F77
C
        INTEGER*2 i
C
        total_used = 0          /* No active.
        next_free = 1           /* Try this one first.
C
        DO 10 i = 1, pool_size  /* For all of them:
        in_use(i) = .FALSE.     /* Not in use,
10      vc_id(i) = 0            /* so no known number.
        RETURN
        END
C GETVC - returns whether the VC block is available or not.
C
        LOGICAL*2 FUNCTION GETVC(net_vc, this_index)
C
$INSERT *>MULTI_VC.INS.F77
C
        INTEGER*2 net_vc, this_index, i
C
        IF (total_used .LT. pool_size) THEN
```

```
C
        getvc = .TRUE.              /* Indicate success.
        this_index = next_free      /* Tell caller allocated slot.
        in_use(next_free) = .TRUE.
        vc_id(next_free) = net_vc
        CALL orgstamp(next_free)    /* Set start of life.
C
C Now prepare for the next allocation call.
C
        total_used = total_used + 1
        IF (total_used .LT. pool_size) THEN
           DO 10 i = 1, pool_size
           IF (in_use(i)) GO TO 10
           next_free = i
           RETURN
10         CONTINUE
        ELSE
           next_free = 0
           RETURN
        ENDIF
C
      ELSE
         getvc = .false.    /* Indicate that the pool is fully used.
      RETURN
      ENDIF
      END
C FREEVC - return a VC block to the unused state.
C
      SUBROUTINE FREEVC(index)
C
$INSERT *>MULTI_VC.INS.F77
C
      INTEGER*2 index
C
      in_use(index) = .FALSE.       /* Free it.
C
C If pool WAS fully used, this must become next to use.
C
      IF (total_used .EQ. pool_size) next_free = index
C
      total_used = total_used - 1   /* Out of used count.
      RETURN
      END
C ORGSTAMP - set origin time for a VC (next full minute).
C
      SUBROUTINE ORGSTAMP(index)
C
$INSERT *>MULTI_VC.INS.F77
C
      INTEGER*2 index, arr(4)
      INTRINSIC INTS
      EXTERNAL TIMDAT
C
      CALL TIMDAT(arr, INTS(4))
```

```
      zero_time(index) = arr(4) + 1
      RETURN
      END
C AGE - return lifelength since origin time for a VC.
C
      INTEGER*2 FUNCTION AGE(index)
C
$INSERT *>MULTI_VC.INS.F77
C
      INTEGER*2 index, arr(4)
      INTRINSIC INTS
      EXTERNAL TIMDAT
C
      CALL TIMDAT(arr, INTS(4))
      age = arr(4) - zero_time(index)
      RETURN
      END
```

A Common Network Error Message Routine:

```
C NEPR.F77, routine to print network status arrays.
C
C This routine gives a formatted output for various
C network status arrays. Its action depends on the number
C of words in the array.
C
C Word 1 translated to XS$xxx
C Word 2 given in decimal and split on CC$XXX/CD$XXX.
C Word 3 given in decimal
C
      SUBROUTINE NEPR(array, no_words)
C
      INTEGER*2 array(1), no_words
C
$INSERT *>fsx_data.ins.f77
C
      INTRINSIC INTS, LT, RT
      EXTERNAL TNOUA, TODEC, TOOCT, TONL
C
      INTEGER*2 cdvalue(6), i
      CHARACTER*6 xsname(-1:14), cdname(6)
      DATA xsname/'XS$NET','XS$CMP','XS$IP','XS$BVC','XS$BPM',
     +      'XS$CLR','XS$RST','XS$IDL','XS$UNK','XS$MEM',
     +      'XS$NOP','XS$ILL','XS$DWN','XS$MAX','XS$QUE','XS$FCT'/
      DATA cdvalue/CD$SHR, CD$LNG, CD$EOU, CD$TMO, CD$RST, CD$NVC/
      DATA cdname/'CD$SHR','CD$LNG','CD$EOU','CD$TMO','CD$RST',
     +      'CD$NVC'/
C
      IF (no_words .LT. 1 .OR. no_words .GT. 3) RETURN
C
      PRINT 9000, xsname(array(1))
9000  FORMAT ('Returned status code: ',A6)
```

```
      IF (no_words .EQ. 1) RETURN
C
      CALL TNOUA('Second word (dec): ', INTS(19))
      CALL TODEC(array(2))
      CALL TONL
      CALL TNOUA('            (as CC$/CD$): ', INTS(26))
      CALL TOOCT(LT(array(2), 8))
      CALL TNOUA('- ', INTS(1))
      CALL TOOCT(RT(array(2), 8))
      DO 10 i = 1,6
      IF (RT(array(2),8) .EQ. cdvalue(i)) THEN
         CALL TNOUA(' (', INTS(2))
         CALL TNOUA(cdname(i), INTS(6))
         CALL TNOUA(') ', INTS(1))
      ENDIF
10    CONTINUE
      CALL TONL
      IF (no_words .EQ. 2) RETURN
C
      PRINT 9010, array(3)
9010  FORMAT ('Third word (dec): ',I5)
      RETURN
      END
```

16

IPCF Programming Strategy

INTRODUCTION

Different applications have different requirements for data transfer. For each application, the designer will define a sequence of messages between the two communicating programs that ensures both programs' proper operation, for error-free communication as well as for various kinds of malfunctions. To achieve this, the designer can, for example, set the X.25 Q-bit on data packets (refer to the description of X$TRAN in Chapter 14), send interrupts, and clear the virtual circuit with appropriate diagnostics. Messages can also be transferred during call setup, by use of the fast select call option. (Refer to the X$F routine descriptions in Chapter 14.)

Usually, a user's IPCF application functions as a front-end program: it acts as a communications link, performing line control, message handling, code conversion, and error control. The front end program tries to establish a virtual call to a program already running on a remote system. That remote program is a server, and is likely to run as a phantom user process.

This chapter discusses some of the techniques and principles of IPCF programming. See Appendix A on X.25 programming guidelines for additional information. The following topics are presented in this chapter.

- Front-end principles
- Server principles

- Performance aspects
- Window and packet sizes in virtual circuits
- Checking return codes
- Network event waiting
- Virtual circuit clearing
- Program closedown
- The effect of START_NET and STOP_NET on IPCF programs

FRONT-END PRINCIPLES

A good front-end design should

- Always keep the user updated on communication progress
- Ensure that user actions cannot cause malfunction of the server

This means that the user program should recognize situations like no server available, the remote system down, and unexpected clear requests in the middle of message exchanges. Also, it should prevent the user from leaving an unterminated virtual circuit alive by just breaking out of the front-end program.

SERVER PRINCIPLES

You can use two main design principles for server design. You might want one server to handle several requests actively in parallel. This design is called a multi-task single server. You must ensure that all active tasks are handled without hanging other tasks and locking up the server. Alternatively, you may want to design a single-threaded server that performs only one request at a time, but usually runs in multiple invocations.

A single-threaded server assigns its port for one call request each time, and reassigns it after completion of a transaction. Obviously, if multiple servers run, they all assign the same port, and they deal with incoming call requests through the scheduling mechanism of the assignment queue. In contrast, a multi-task server assigns one or several ports for an infinite number of calls, and regularly scans for new incoming call requests.

Whichever principle you choose, there is a maximum value for active requests, limited by either the 'parallel' capacity of the multi-task server or by the number of running single-threaded servers. Your design has to cope with how front-end programs are handled when there is no server-handling available.

A convenient solution to the "no available server" problem for multiple single-threaded servers is to run a special server, that assigns the common server port for an infinite number of calls at the end of the assign queue (see the description of X$ASGN). This special server will be called only if no other server is available. Its sole action is to transmit the message "No server available," and clear the call.

The single-threaded server will have a code path corresponding to the designed sequence of messages. A multi-task server will need status-checking loops for all active tasks, including time-out actions to prevent virtual circuits from hanging. Therefore, in general, the single-threaded server should be easier to design and maintain.

A well-designed server should be stable if it encounters errors. Preferably it should not crash but rather reinitialize itself, and reenter service. The condition handling mechanism of PRIMOS should be used to catch and properly handle forced logouts and similar events. The server should also run some sort of log file, so that abnormal behavior can be traced back later.

An application based on multiple servers must be designed to not depend on any user-specific property such as the local user number of the server. The reason for this is that you can never know which server is used for a specific transaction.

PERFORMANCE ASPECTS

Although the IPCF interface to PRIMENET functions independently of transmission media such as RINGNET or a PDN synchronous link, the throughput varies greatly between these. An application that runs well over RINGNET, due to the very high throughput over the ring, may turn out to be very slow over a synchronous line. In general, application designs should

- Minimize the amount of data transferred over the network
- Keep the number of messages at the user level as low as possible
- Keep the number of protocol turnarounds (when one of the programs has to wait for the other to send a message back before it can proceed) as small as possible

The message transfer structure of X$TRAN and X$RCV makes the partitioning of messages into proper X.25 protocol data packets invisible for the application programmer. However, each message that is handed to X$TRAN will be packetized inside PRIMENET. Thus, message sizes should be made to match the used packet size, or integer multiples thereof.

WINDOWS AND PACKET SIZES IN VIRTUAL CIRCUITS (THROUGHPUT)

Your application should always ensure that PRIMENET sees sufficient user-provided buffer space to handle incoming data.

Further, you can optimize program performance by specifying a large window and packet size for the virtual circuit. This reduces overhead, since fewer packets have to be analyzed and handled, and PRIMENET can have more packets outstanding.

You can enlarge the window and packet size in two ways. One way is to use the XK$FCT key for the XLCONN call that establishes the virtual call. Using XK$FCT is straightforward, and means that PRIMENET adds a predefined facility field to the call. This field has one value for direct Prime-to-Prime links, and other values for links through PDNs. The actual value will vary with the PDN, and in some cases it will be the X.25 default value. The only risk with this strategy is running an international link over multiple PDNs. If your local PDN link increases either window or packet size, the international gateway to the next PDN might reject the packet facility request, and clear your call attempt.

The other way to increase the window and packet size is to provide a facility field. You have to check on any restrictions on the flow control negotiation facilities that are implied by your PDN subscription. If you require maximum window and packet sizes as defined within X.25, your parameter value for a direct Prime-to-Prime link will be reduced to the maximums actually supported over the medium.

The receiving server should not take any specific steps, since a call accept without facility field grants the caller's required window and packet size. If the accept packet explicitly requires something else, it is only permitted to negotiate values closer to the X.25 default.

Packet size increases will prove useful only if the message buffers given to X$TRAN are sufficiently large; preferably, the message length should be an integer multiple of the packet size used.

NETWORK EVENT WAITING

One essential feature of IPCF applications is their asynchronous nature. Your user program initiates data transfers and other network activities, but returns from many of IPCF subroutines immediately after the request is launched, rather than after it completes. This is to your advantage. Your program can perform other functions while PRIMENET is working for you.

Some return codes indicate serious errors or an inability to initiate the requested actions. For example, XS$BPM means that your call arguments contain illegal values. Similarly, the return code XS$MEM indicates work contention inside your local PRIMENET, meaning that PRIMENET has temporarily run out of buffers. These kinds of codes are returned immediately.

In contrast, IPCF routines in due course return the result of network actions in status arrays, and at the same time notify your network semaphore. This enables your program to wait on the network semaphore as soon as it is idle, and to find out what has happened by checking the status arrays, once it wakes up from the network semaphore. The X$WAIT function waits on the network semaphore. It allows you to require either an infinite wait, or a combined timeout/network-event wait. In the latter case, the returned function value will indicate if you woke up on timeout or on an actual event.

Semaphores usually include event counters that monitor the number of waiting processes or events to be handled. The PRIMENET semaphore differs from that principle. It generates only one collective notification for all your network events, until you wait on the semaphore again by invoking X$WAIT (and thus immediately awake). This is to avoid the problems of overnotification in case the IPCF user program does not wait on the semaphore. Therefore, if your application has several outstanding network requests, such as multiple supplied receive buffers, you should check the status for all of them, not only the first one, before waiting again on the network semaphore, or your program might hang.

An infinite waiting time is probably convenient when you have assigned a port and are waiting for the next call to come in. As a caller, you could also wait infinitely, since PRIMENET times-out the call if it is not accepted properly, and wakes you up.

Once a virtual circuit has passed into data transfer phase, there are no time-out mechanisms inside PRIMENET for an idle circuit with no data to transfer. If one side does an X$RCV, followed by an infinite wait for the other end to respond, and the other end 'crashes' without clearing the virtual circuit before transmitting, the receiving side actually hangs forever. It would be better to have a (long) time-out and give the user terminal a message about possible problems.

CHECKING RETURN CODES

Calls to X$CLRA and X$UASN always return without error; all other calls return with (or affect the value of) a status word or array. The 'immediate-return' design of the IPCF routines implies that several return status values are 'reasonable', so normally an IPCF application program will have several 'reasonable' code paths following each IPCF call.

For example, when an application has reached the data transfer phase, and is manipulating a number of calls to X$IRAN and X$RCV, the return statuses XS$IP, XS$CMP, possibly XS$RST, and XS$CLR are all 'normal', and must be handled. This means that there might be a sequence of statements like

```
IF (returned_status .EQ.  XS$xxx) GO TO yyy
```

In the above case, it is essential that a local copy of returned status be taken and used in the sequence of IF statements, to guarantee that the branching is correct, and not upset by a sudden change of the returned status value, done by PRIMENET, in the middle of the testing. The most usual change would obviously be the transition from XS$IP to some other status, indicating that the operation terminated.

Some sample code showing this follows.

```
50    CALL X$WAIT(10)                 /* Idle a while...
      J = VCSTAT(1)                   /* Copy circuit status value.
      IF (J .EQ. XS$IP) GOTO 50       /* Keep idling...
      IF (J .EQ. XS$CLR) GOTO 10      /* OK to accept next call.
      IF (J .EQ. XS$CMP) GOTO 50      /* Wait for clear.
```

VCSTAT(1) is copied into J, and the three comparisons are made against J, rather than VCSTAT(1), ensuring that the value does not change between the tests.

The return code XS$BVC occurs when you order actions on a virtual circuit that you do not control. It is basically a fatal error code, implying that your program is in error. However, it can sometimes also be a 'normal' (non-fatal) return code for calls to X$IRAN, X$RCV, X$RSET, and X$CLR. This occurs if the virtual circuit has been cleared by the other side or by the network, after your last status test (which did not return XS$CLR), but before your call to any of these routines.

IPCF applications should always check the status array returned or affected by any IPCF subroutine call. Failure to check errors could lead to difficulties, such as are shown in this example.

```
NPORT = 3279
CALL X$ASGN(NPORT, 1, NSTAT)
CALL X$WAIT(0)
```

Because the port number 3279 is invalid, X$ASGN returns with the error XS$BPM in NSTAT. The example fails to discover the error, and waits for a network event on an unassigned port. It will wait forever.

VIRTUAL CIRCUIT CLEARING

The X.25 standard states explicitly that the fate of packets in transmission when either side requests a circuit CLEAR is undefined. They can either be delivered properly or dropped. The consequence of this for application programs is that the clear should normally be done by the side that receives the last message. It is not sufficient to wait for X$TRAN to yield XS$CMP before you call X$CLR; the transmitted message can still get dropped by the other side if the clear is detected before the corresponding X$RCV call has had time to reach XS$CMP state.

PROGRAM CLOSEDOWN

When you terminate an IPCF application, you should ensure the following.

- When you have assigned ports for receiving incoming calls, make sure that you call either X$UASN or X$CLRA to release these ports. Otherwise, these ports will remain assigned for you, routing incoming calls to you and preventing other applications from using them to receive calls.

- As long as you have a running virtual circuit, on which you have sent a clear request (by calling X$CLR or any corresponding routine), PRIMENET will suddenly write into your virtual circuit status array, to indicate that the remote end has confirmed the clear request. To prevent overwriting of other programs that may be executed later, you should not 'CALL EXIT' or return to command level until you do one of the following.

    - All virtual circuit status arrays' first words have changed to XS$CLR, indicating that the clear request is confirmed.

    - You have called X$CLRA, which forces an immediate drop of all your virtual circuit references. (PRIMENET will still handle the clear confirmation properly.) The call to X$CLRA will generate a clear request with diagnostic byte 0, if the circuit has not already been cleared.

Note

If you request a clear immediately for an incoming call, without first accepting it, no virtual circuit status array is created, so you cannot detect the confirmation.

THE EFFECT OF START_NET AND STOP_NET ON IPCF PROGRAMS

At PRIMENET rev 19.3 it becomes possible to start and stop the NETMAN process, without coldstarting PRIMOS, through the START_NET and STOP_NET commands. Therefore, if your applications run under PRIMENET 19.3, locally or remotely, they should have logic to deal with what happens when the network is stopped or started. This is especially important for servers running as phantoms on nodes that execute STOP_NET.

When NETMAN is stopped, active virtual circuits are cleared. A new Prime-defined clearing diagnostic, CD$NSV, indicates this event. Any IPCF application that receives this clearing diagnostic will know that either its local NETMAN or the remote node's NETMAN has been closed down.

When START_NET is invoked, the whole local network configuration database is reinitialized. Among other things, this means that all previous port assignments are wiped out. Any server waiting infinitely on an 'old' port assignment is left hanging, and never awakens.

One strategy here would be to not wait infinitely after assigning a port, but to wake up periodically. Any inquiry by X$GCON/X$FGCN/XLGCON would then indicate network down by returning XS$NET, and the server could take appropriate action. Alternatively, the System Administrator should have all servers log out before issuing STOP_NET.

The "Network not running" status, XS$NET, is returned only by IPCF routines that initiate network connections, such as X$ASGN; X$CONN, X$FCON, XLCONN; X$GCON, X$FGCN, XLGCON; and by the status routine X$STAT. Other routines combine this status with XS$BVC, which is also returned if you require an action for a virtual circuit that does not belong to you.

17

FTS Programming

INTRODUCTION

The File Transfer Service includes three commands, FTR, FTOP and FTGEN. These commands comprise the user, operator, and System Administrator interfaces to FTS, respectively. In addition, the File Transfer Service provides a program interface. As of Revision 19.3, this interface is implemented in the form of one subroutine, named FT$SUB.

The FT$SUB subroutine allows an application program to perform any function that a user can perform by using the FTR command. A program can use FT$SUB to submit, modify, cancel, abort, hold, release, and check the status of file transfer requests.

This chapter describes:

- How to set up your FORTRAN (FTN), FORTRAN 77 (F77), or PL/I Subset G (PL1G) program and its load sequence to allow the use of FT$SUB.
- How to use the FT$SUB subroutine to submit, control, and determine the status of transfer requests.
- Several sample programs using FT$SUB.

PROGRAM SETUP FOR FT$SUB

The FT$SUB subroutine is designed to be called from FORTRAN (FTN), FORTRAN 77 (F77), and PL/I Subset G (PL1G) programs. FT$SUB cannot be called from R-mode or S-mode programs.

The FT$SUB subroutine makes use of FTS-specific keys and error codes, all of which have names beginning with F$ or Q$. These keys and codes are defined in an insert file in SYSCOM. In addition, the FT$SUB subroutine is installed as a shared subroutine library. A library file is used to satisfy the references to this subroutine during your program load sequence.

To use the FT$SUB subroutine, you must

- Declare the FT$SUB subroutine in your program, if it is a PL/I Subset G program
- Use an %INCLUDE or $INSERT statement in your program to define the keys and error codes related to FT$SUB
- Use the LIBRARY VFTSLB command in your program load sequence to load the FTS library

When the above steps are performed, you can invoke FT$SUB in your program, as described in the section entitled INVOKING THE FT$SUB SUBROUTINE.

Declaring FT$SUB

You must use the following declaration for FT$SUB in PL/I Subset G programs.

```
dcl ft$sub entry(fixed bin(15),char(32) var,char(32) var,
          char(*) var,char(*) var,char(255) var,char(32) var,
          fixed bin(15),ptr,fixed bin(15),ptr,fixed bin(15),
          fixed bin(15));
```

The full calling sequence, rarely needed, is as follows.

```
call ft$sub(key,request_name,internal_name,user_cmdl,prog_cmdl,
           '',queue,user_query,addr(request_data),l,
           addr(error_data),l,code);
```

Specific calling sequences needed for particular uses of FT$SUB are described in the section that discusses each such use.

Defining Keys and Error Codes

To allow your program to represent FTS-specific numeric values for keys and error codes, you must include in your program a statement that defines the appropriate FTS user's file. The statement to be used depends on the language in which you are writing the program.

For a PL/I Subset G (PL1G) program, use the following statement.

```
%INCLUDE 'SYSCOM>FT$SUB.INS.PL1';
```

For a FORTRAN 77 (F77) program, use the following statement.

```
%INCLUDE 'SYSCOM>FT$SUB.INS.FTN'
```

For a FORTRAN (FTN) program, use the following statement.

```
$INSERT SYSCOM>FT$SUB.INS.FTN
```

Each statement will cause the specified file to be logically included in your program. Each file contains a list of statements that define constants, or keys, that you will use when calling FT$SUB.

Loading the FTS Subroutine Library

Place the following command immediately prior to the final LIBRARY command in your program load sequence.

```
LIBRARY VFTSLB
```

For example, the load sequence for a PL1G program named SEND_FILE might be as follows.

```
SEG -LOAD
LOAD SEND_FILE
LIBRARY PL1GLB
LIBRARY VFTSLB
LIBRARY
MAP 3
MAP SEND_FILE.MAP
SAVE
QUIT
```

See the SEG and LOAD Reference Guide for more information on program load sequences.

INVOKING THE FT$SUB SUBROUTINE

The FT$SUB subroutine allows eight different functions to be performed for any given invocation. These transfer request functions are as follows.

- Submittal
- Parameter modification
- Canceling
- Aborting
- Holding
- Releasing
- Status retrieval of a user's requests
- Status retrieval of all requests on the system

These functions are distinguished by the first parameter of the FT$SUB subroutine. That parameter is a fixed bin(15) value. There are eight legal values for the argument, corresponding to the eight functions.

This section describes the categorization of these functions, and fully describes the functions themselves. It then describes the use of internal vs. external names. Finally, this section describes the following information returned by FT$SUB.

- The error code
- The request data structure
- The error data structure

Note

In general, user processes can operate only on their own submitted requests. There are two exceptions, however. First, a user process with the login ID SYSTEM can operate on any request. Second, any process can request status and parameter information for any request on the system, and receive limited information about that request.

Function Categories

The eight functions of FT$SUB may be logically grouped into four categories. The first and second functions, submission and parameter modification, belong to their own categories — submission and modification. The next four functions belong to a category called status change operations, because they change only the status of a request. The final two functions belong to a category called status retrieval operations, because their purpose is to retrieve information about an existing transfer request, without changing the request itself.

Submission: The first function is transfer request submission. This is the only function that adds a new transfer request; the other functions operate on existing requests. The FTR command uses this function when it is submitting a new request. The parameters for the request are specified through two character strings that contain command line options.

Modification: The second function is transfer request modification. It is used by the FTR command when the -MODIFY option is specified. The parameters to be changed are specified through two character strings that contain command line options.

Status Change Operations: The next four functions, which are the cancel, abort, hold, and release functions, involve the modification of the status of a transfer request. The FTR command uses these functions when the -CANCEL, -ABORT, -HOLD, or -RELEASE options are specified.

Status Retrieval Operations: The final two functions obtain the status and parameter information for a request. The FTR command uses these functions when the -STATUS or -STATUS_ALL options are specified. One function obtains full information for a transfer request from the user who calls FT$SUB, and the other function obtains partial information for any transfer request on the system.

Transfer Request Submission

This function performs the initial submission of a transfer request. It is similar in effect to the command, "FTR pathname". To submit a transfer request, use the following calling sequence.

```
call ft$sub(f$subm,'',internal_name,user_cmdl,prog_cmdl,'','',
            0,addr(request_data),1,addr(error_data),1,code);
```

The following table consists of arguments that are passed to the FT$SUB subroutine as input parameters.

Input Arguments	Meaning
F$SUBM	This key specifies that a submission operation is to be performed.
'','','',0	Three null strings and a 0 stand for arguments that are not used by FT$SUB during a submission operation. You must pass these arguments exactly as shown, or FT$SUB will return an error code of E$BPAR.
internal_name,	Set to null on the initial call. Output after submission.
user_cmdl, prog_cmdl	These strings specify the command lines for the user and program. These command lines provide the details of the submission operation to FT$SUB.
addr(request_data),1	These arguments are a pointer to a request information structure (addr(request_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to reflect the results of the submission. This structure is described fully in the section The Request Data Structure, below. If you do not want to provide a request data structure, specify the address and the version number of the structure as null(),0. Any other combination of settings for these parameters will result in the error code E$BPAR being returned.
addr(error_data),1	These arguments are a pointer to an error information structure (addr(error_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to provide extra information in case an error occurs during the submission. This structure is described fully in the section Error Data Structure, below.

If you do not want to provide an error data structure, specify the address and the version number of the structure as null(),0.

Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

There are two output arguments whose values are modified by FT$SUB for use by the calling program.

Output Arguments	Meaning
internal_name	The internal name of the submitted request. This value is returned only if the returned error code is 0. You should output this field to the user after the submission operation, as does the FTR command.
code	The error code that represents the success or failure of the operation. This error code may be a standard PRIMOS file system error code, or may be an FTS-specific error code.

Setting Up for Submission: Before calling FT$SUB, initialize user_cmdl and prog_cmdl to contain the user and program command lines, or pass constant strings, as appropriate.

The contents of user_cmdl are expected to be one or two pathnames followed by a list of options. Although the intent of user_cmdl is to contain a command line with options as specified by a user, this is not a requirement. For example, the program may construct user_cmdl by allowing the user to select choices on a menu, with the program adding options for each choice.

The intent of prog_cmdl is to allow the program to provide recommendations for options should the user not specify them. Typical examples include the specification of -NO_QUERY, the setting of the -COPY and -NO_COPY switches, and the destination site (-DST_SITE). Whereas user_cmdl contains the source and destination pathnames, prog_cmdl can contain only options. The options specified in user_cmdl will override corresponding specifications in prog_cmdl.

For example, suppose user_cmdl and prog_cmdl are set to the following values.

```
user_cmdl: 'IMPORTANT.MEMO JONES>IN_TRAY>MEM.AKB/001 -SRC_NTFY
           -LOG MY.LOG'

prog_cmdl: '-DSTN_SITE BIRCH -NO_COPY -NO_QUERY
           -LOG USER_REQUESTS>REQUEST.LOG'
```

This will result in a transfer request for the file IMPORTANT.MEMO to be copied into JONES>IN_TRAY>MEM.AKB/001 on the system named BIRCH (-DSTN_SITE). No temporary copy of the file will be made on the local node (-NO_COPY). The requesting user will be notified of the start and end of the transfer (-SRC_NTFY). A request log file called MY.LOG is also created.

Notice how -LOG MY.LOG in the user_cmdl overrides the -LOG USER_REQUESTS>REQUEST.LOG in prog_cmdl. The request is submitted with another non-default option in force, which is the -NO_COPY option. This is becuase the option was present in prog_cmdl. If user_cmdl is specified by a user, and prog_cmdl is always provides by the program as shown, then the user need not specify -NO_COPY. It is esentially the default specification for this program.

Note

You should always include the -NO_QUERY option in prog_cmdl to suppress user queries during request submission. See the description of the FTR -COPY option in Chapter 6 for an example of such a query. The default is to query the user.

Error Recovery: The following table shows the FTS error codes that can be returned with submission. See also the section below on Error Codes for a description of general error codes.

Submission Error Codes

F$BDCL	Bad command line format.
F$BDDN	Bad device name.
F$BDKW	Unknown keyword.
F$BDSN	Bad site name format.
F$CNOP	Conflicting option.
F$CPLS	Copy option applies only to local source file.
F$DFNS	Destination file has not been specified.

F$DLLS	Delete option applies only to local source file.
F$DRNA	Device transfer from remote site is not allowed.
F$DSNC	Destination site is not configured.
F$DUIN	Destination user name invalid.
F$DUNS	Destination user is not specified when destination notify requested.
F$DUOP	Duplicate option.
F$FPTL	Full pathname too long.
F$IDFT	Invalid destination file type.
F$IFDC	Illegal file or directory conversion.
F$INMS	Invalid message level.
F$ISFT	Invalid source file type.
F$MCLP	Missing command line parameter.
F$MBNL	Message level specified but request log treename has been omitted.
F$NCLS	No copy option applies only to local source file.
F$NDLS	No delete option applies only to local source file.
F$NTCF	Not configured.
F$PINS	Segment directory transfer to and from a Rev 1 site is not supported.
F$PSFQ	Passworded pathname must be fully qualified.
F$RLST	Request log treename same as source or target treename !!
F$RTIS	Remote treename incorrectly specified.
F$SDSL	Source or destination site must be local.
F$SFNE	Source file does not exist.
F$SFNS	Source file has not been specified.
F$SFTD	Specified and actual source file types differ.
F$SSNC	Source site is not configured.
F$SUIN	Source user name invalid.

F$SUNS Source user not specified when source notify was requested.

F$TDFN Transfer to a device as well as a file is not allowed.

F$TDNS Transferring a SEG directory to a device is not supported.

F$TFNP Transferring a file to itself is not possible.

F$UNOP Unknown option.

Q$FULL Queue full.

Q$QBLK Queue blocked.

Q$QNEX Queue does not exist.

Q$UCTF Unable to create temporary file.

Example: The following example shows a simple use of FT$SUB for request submission. No declarations are provided, since they are described above and in the Subroutines Reference Guide.

```
call tnoua('Enter command line: ',20);
call cl$get(command_line,160,code);
if code^=0 then return;

call ft$sub(f$subm,'',internal_name,command_line,
            '-SRC_NTFY -NO_COPY','','',0,null(),0,null(),0,
            code);
if code^=0 then return;

call tnoua('Your request is #',17);
call tnou(internal_name,length(internal_name));
return;
```

Transfer Request Modification

This function is used to change one or more parameters of a transfer request. The request must have already been submitted. This is similar in function to the "FTR -MODIFY name" command. To modify a transfer request, use the following calling sequence.

```
call ft$sub(f$mdfy,request_name,internal_name,user_cmdl,prog_cmdl,
            '',queue,0,addr(request_data),1,addr(error_data),1,
            code);
```

The following table shows arguments that are passed to the FT$SUB subroutine as input parameters.

Input Arguments	Meaning
F$MDFY	This key specifies that a modification operation is to be performed.
request_name	This argument contains the internal or external name of the request to be modified.
internal_name	This argument contains a null string during the initial call. During subsequent calls, this argument contains either a null string or the internal name of a request from which point in the queue scanning is to start. See the section entitled Internal vs. External Names, below, for more information.
user_cmdl, prog_cmdl	These strings specify the options that are to modify the request.
'',0	A null string and a 0 stand for arguments that are not used by FT$SUB during a modification operation. You must pass these arguments exactly as shown, or FT$SUB will return an error code of E$BPAR.
queue	This string specifies the name of the FTS queue to search for the request. If all queues are to be searched, pass the null string in queue.
addr(request_data),1	These arguments are a pointer to a request information structure (addr(request_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to reflect the results of the operation. This structure is described fully in the section The Request Data Structure, below. If you do not want to provide a request data structure, specify the address and the version number of the structure as null(),0. Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

addr(error_data),1	These arguments are a pointer to an error information structure (addr(error_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to provide extra information in case an error occurs during the operation. This structure is described fully in the section The Error Data Structure, below. If you do not want to provide an error data structure, specify the address and the version number of the structure as null(),0. Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

The following are arguments whose values are modified by FT$SUB for use by the calling program.

Input Arguments	Meaning
internal_name	The internal name of the modified request. This value is returned only if the returned error code is 0 or F$TRPR (Transfer in progress). You should output this field to the user after the operation, as does the FTR command.
code	The error code representing the success or failure of the operation. This error code may be a standard PRIMOS file system error code, or may be an FTS-specific error code.

Setting Up for Modification: Before you call FT$SUB, initialize user_cmdl and prog_cmdl to contain the user and program command lines, or pass constant strings, as appropriate. These strings must contain only options, as the source and destination pathnames cannot be changed.

In addition, most applications will set prog_cmdl to the null string, since "background" option specifications are not normally needed during a modify operation. Any options that are specified in prog_cmdl will be overridden by any identical options in user_cmdl.

The following options (and their abbreviations) cannot be specified in either user_cmdl or prog_cmdl, because their corresponding parameters are not changeable.

-COPY	-NO_COPY
-DSTN_SITE	-QUEUE
-DSTN_FILE_TYPE	-SRC_FILE_TYPE
-HOLD	-SRC_SITE

If any of these options are present in user_cmdl or prog_cmdl during a modify operation, an error code will be returned.

Error Recovery: If any problems occur during the modification operation, a non-zero value will be returned in code, and the operation will not take place. Errors fall into one of the following several categories. See also the section on Error Codes below for a description of general error codes and those listed in the section on Error Data Structure.

- Illegal calling sequence. The arguments passed to the subroutine by the calling program are illegal. This can result in the E$BPAR (Bad PARameter) error code being returned if incorrect version numbers are supplied for the request or error data structures, or if other arguments are not supplied as specified in the above description.

- Unrecognized option (error code F$UNOP) or keyword (error code F$BDKW).

- Unable to modify specified parameters. The modify function cannot be used to change certain parameters, described above in the list of illegal options. The following error codes are returned.

Code	Meaning
F$QNMD	Queue name may not be modified.
F$NCMD	NO_COPY flag may not be modified.
F$CPMD	COPY flag may not be modified.
F$SSMD	Source site may not be modified.
F$DSMD	Destination site may not be modified.
F$HDMD	Hold flag may not be modified.
F$SFMD	Source file type may not be modified.
F$DFMD	Destination file type may not be modified.

If FT$SUB cannot locate the request with the request_name and internal_name fields that were passed to it by the calling program, it will return an error code of E$EOF (End of file). Your program should not generate the "End of file" error message after receiving the E$EOF error code from FT$SUB. Instead, it should generate a message such as "Request not found". See the section entitled Internal vs. External Names, below, for more information.

- Insufficient access. If the request belongs to another user, and the user calling FT$SUB is not logged in as user SYSTEM, the error code F$NERF (No eligible request of this name found) is returned.

- Unable to modify the request. Two error codes may be returned if the request is in a state that prevents it from being modified. If the request is being processed, the error code F$TRPR (Transfer in progress) will be returned. If the request has been aborted, the error code F$RQAB (Request already aborting) will be returned.

Example: The following example shows a simple use of FT$SUB for request modification. No declarations are provided, as they are described above and in the Subroutines Reference Guide.

```
call tnoua('Enter request name: ',20);
call cl$get(request_name,32,code);
if code^=0 then return;

call tnoua('Enter command line: ',20);
call cl$get(command_line,160,code);
if code^=0 then return;

internal_name='';
call ft$sub(f$mdfy,request_name,internal_name,command_line,
            '','',0,null(),0,null(),0,code);
if code^=0 then return;

call tnoua('The modified request is #',25);
call tnou(internal_name,length(internal_name));
return;
```

Changing the Status of a Transfer Request

To change the status of a transfer request, use the following calling sequence.

```
call ft$sub(key,request_name,internal_name,'','','',queue,
            user_query,addr(request_data),1,addr(error_data),1,
            code);
```

The table below shows arguments passed to the FT$SUB subroutine as input parameters.

Input Arguments	Meaning
key	This argument specifies one of four operations to be performed on the request: F$CANC to cancel; F$ABRT to abort; F$HOLD to hold; and F$RLSE to release.
request_name	This string contains the internal or external name of the request to be operated upon.
internal_name	This argument contains a null string during the initial call. During subsequent calls, this argument contains either a null string or the internal name of a request from which point in the queue scanning is to start. See the section entitled Internal vs. External Names, below, for more information.
'','',''	Three null strings, for arguments that are not used by FT$SUB during a status change operation. You must pass these arguments exactly as shown, or FT$SUB will return an error code of F$CLMN.
queue	This string contains the name of the FTS queue to search for the request. If all queues are to be searched, pass the null string in queue.
user_query	This argument specifies whether the submitting user is to be queried or not. Set this argument to F$UQRY (value 1) if the user query through terminal I/O is to be performed. Set this argument to F$NQRY (value 0) if no user query is to be performed. Currently, the setting of this argument has no effect.
addr(request_data),1	These arguments are a pointer to a request information structure (addr(request_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to reflect the results of the operation. This structure is described fully in the section The Request Data Structure, below.

If you do not want to provide a request data structure, specify the address and the version number of the structure as null(),0.

Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

addr(error_data),1

These arguments are a pointer to an error information structure (addr(error_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to provide extra information in case an error occurs during the operation. This structure is described fully in the section The Error Data Structure, below.

If you do not want to provide an error data structure, specify the address and the version number of the structure as null(),0.

Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

The following are arguments whose values are modified by FT$SUB for use by the calling program.

Output Arguments	Meaning
internal_name	The internal name of the affected request. This value is returned only if the returned error code is 0 or one of several possible error codes described below. You should output this field to the user after the operation, as does the FTR command.
code	The error code representing the success or failure of the operation. This error code may be a standard PRIMOS file system error code, or may be an FTS-specific error code.

Error Recovery: If any problems occur during the operation, a non-zero value will be returned in code, and the operation will not take place. These errors fall into one of several categories. See also the section on Error Codes below for a description of general error codes.

- Illegal calling sequence. The arguments passed to the subroutine by the calling program are illegal. This can result in the E$BPAR (Bad PARameter) error code being returned if incorrect version numbers are supplied for the request or error data structures, or if other arguments are not supplied as specified in the above description.

- Command lines not null. If your program does not pass null strings as the fourth and fifth arguments to FT$SUB, the error code F$CLMN (Command Lines Must be Null) will be returned.

- Unable to find specified request in data base. If FT$SUB cannot locate the request with the request_name and internal_name fields that were passed to it by the calling program, it will return an error code of E$EOF (End of file). Your program should not generate the "End of file" error message after receiving the E$EOF error code from FT$SUB. Instead, a message such as "Request not found" should be generated. See the section entitled Internal vs. External Names, below, for more information.

- Insufficient access. If the request belongs to another user, and the user calling FT$SUB is not logged in as user SYSTEM, the error code F$NERF (No eligible request of this name found) is returned.

- Unable to change the status of the request. Several error codes may be returned if the request is in a state that prevents its status from being changed. These error codes are:

Code	Meaning
F$TRPR	Transfer in progress
F$RQHU	Request already put on hold by user
F$RQHO	Request already put on hold by operator
F$RQHF	Request already put on hold by FTS
F$RQAB	Request already aborted
F$RQWT	Request waiting
F$RHPR	Request held by operator

The first five codes are self-explanatory. The sixth code, F$RQWT, is produced when an attempt is made to release a request that has not been placed on hold. The last code, F$RHPR, is produced when an attempt is made by a user not logged in as SYSTEM to release a request placed on hold by an operator (user SYSTEM).

Example: The following example shows a simple use of FT$SUB for changing the status of a request. No declarations are provided, since they are described above or in the Subroutines Reference Guide.

```
call tnoua('Enter request name: ',20);
call cl$get(request_name,32,code);
if code^=0 then return;

call tnou('Choose one of the following options:',36);
call tnou('',0);
call tnou('   1. Cancel the request',24);
call tnou('   2. Abort the request',23);
call tnou('   3. Hold the request',22);
call tnou('   4. Release the request',25);
call tnou('   5. Exit this program',23);
call tnou('',0);
call tnoua('Enter your choice: ',19);
call cl$get(command_line,160,code);
if code^=0 then return;

     if command_line='1' then key=f$canc;
else if command_line='2' then key=f$abrt;
else if command_line='3' then key=f$hold;
else if command_line='4' then key=f$rlse;
else if command_line='5' then return;
else do;
     call tnou('Illegal response.',17);
     code=e$ivcm;  /* Invalid command. */
     return;
     end;

internal_name='';
call ft$sub(key,request_name,internal_name,'','','','',0,
          null(),0,null(),0,code);
if code^=0 then return;

call tnoua('The affected request is #',25);
call tnou(internal_name,length(internal_name));
return;
```

Status Retrieval of a Transfer Request

To retrieve the status of a particular transfer request, use the following calling sequence.

```
call ft$sub(key,request_name,internal_name,'','','',queue,0,
            addr(request_data),1,addr(error_data),1,code);
```

The information on the request will be returned in the request_data structure, described below. The arguments below are passed to the FT$SUB subroutine as input parameters.

Input Arguments	Meaning
key	This argument specifies one of two forms of status retrieval: F$STAT to retrieve complete status and parameter information on a transfer request that belongs to the user calling FT$SUB; and F$STAL to retrieve partial status and parameter information on any transfer request on the system.
request_name	This string contains the internal or external name of the target request.
internal_name	This argument contains a null string during the initial call. During subsequent calls, this argument contains either a null string or the internal name of a request from which point in the queue scanning is to start. See the section entitled Internal vs. External Names, below, for more information.
'','','',0	Three null strings and a 0 stand for arguments that are not used by FT$SUB during a status retrieval operation. You must pass these arguments exactly as shown, or FT$SUB will return an error code of F$CLMN.
queue	This string contains the name of the FTS queue to search for the request. If all queues are to be searched, pass the null string in queue.
addr(request_data),1	These arguments are a pointer to a request information structure (addr(request_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to return the status information of the request. This structure is described fully

in the section The Request Data Structure, below.

Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

addr(error_data),1

These arguments are a pointer to an error information structure (addr(error_data)) followed by the version number of that structure (1). Although the pointer itself is an input argument to FT$SUB, the structure it points to is modified by FT$SUB to provide extra information in case an error occurs during the operation. This structure is described fully in the section The Error Data Structure, below.

If you do not want to provide an error data structure, specify the address and the version number of the structure as null(),0.

Any other combination of settings for these parameters will result in the error code E$BPAR being returned.

FT$SUB modifies the values of the arguments below for use by the calling program.

Output Arguments	Meaning
internal_name	The internal name of the target request. This value is returned only if the returned error code is 0. If your program is retrieving status information for display purposes, you should output this field to the user along with other status and parameter information, as does the FTR command.
code	The error code representing the success or failure of the operation. This error code may be a standard PRIMOS file system error code, or may be an FTS-specific error code.

Error Recovery: If any problems occur during the status retrieval operation, a non-zero value will be returned in code, and the operation will not take place. These errors fall into one of several categories. See also the section on Error Codes below for a description of general error codes.

- Illegal calling sequence. The arguments passed to the subroutine by the calling program are illegal. This can result in the E$BPAR (Bad PARameter) error code being returned if incorrect version numbers are supplied for the request or error data structures, or if other arguments are not supplied as specified in the above description.

- Command lines not null. If your program does not pass null strings as the fourth and fifth arguments to FT$SUB, the error code F$CLMN (Command Lines Must be Null) will be returned.

- Unable to find specified request in data base. If FT$SUB cannot locate the request with the request_name and internal_name fields passed to it by the calling program, it will return an error code of E$EOF (End of file). Your program should not generate the "End of file" error message after receiving the E$EOF error code from FT$SUB. Instead, it should generate a message such as "Request not found". See the section entitled Internal vs. External Names, below, for more information.

- Insufficient access. If the request belongs to another user, and the user calling FT$SUB is not logged in as user SYSTEM, the error code F$NERF (No eligible request of this name found) is returned. This error code is returned only if key is F$STAT.

Example: The following example shows a simple use of FT$SUB for retrieving the status of a request. No declarations are provided, as they are described above or in the Subroutines Reference Guide.

```
call tnoua('Enter request name: ',20);
call cl$get(request_name,32,code);
if code^=0 then return;

internal_name='';
call ft$sub(f$stat,request_name,internal_name,'','','','',0,
          addr(request_data),1,null(),0,code);
if code^=0 then return;

call tnoua('The request is #',25);
call tnou(internal_name,length(internal_name));

call tnoua('Last attempt was '||request.last_date||' at ',29);
call tovfd$(divide(request.last_time,60,15));
call tnoua(':',1);
if mod(request.last_time,60)<10 then call tnoua('0',1);
call tovfd$(mod(request.last_time,60));
call tnoua(', ',2);
```

```
if request.ntry=1 then call tnou('one connection attempt.',23);
   else do;   /* if request.ntry^=1 */
        call tovfd$(request.ntry);
        call tnou(' connection attempts.',21);
        end;  /* if request.ntry^=1 */

return;
```

Internal vs. External Names

For operations other than request submission, the FT$SUB subroutine is designed to allow one operation on one file transfer request per invocation. In some cases, you may want a calling program to perform operations over a set of transfer requests. Quite often, this set of requests constitutes one of the following.

- All of the requests in the system
- All of the requests that belong to the calling user
- All of the requests that have a certain external name that belong to the calling user

In each of these cases, FT$SUB provides a simple way of calling programs to invoke it iteratively for all of the transfer requests in the set. You do this by providing certain information to the calling program upon the completion of an FT$SUB function that can then be recycled by the program into another call to FT$SUB.

In addition, the calling program may further limit the set to only those requests in a specific FTS queue, by providing a non-null queue argument.

Calling Sequence Support for Iteration: In each of the calling sequences described for FT$SUB except for the submission calling sequence, there are two arguments, request_name and internal_name. These are the second and third arguments of the FT$SUB calling sequence.

A request name can be the name of the file being transferred, which is the request_name, or the number of the request that is provided by FTS. The request number is the internal name of a request.

The request_name argument is an input-only argument. The calling program must fill this field with an identifier of the transfer request it is referencing. It can do this in one of three ways.

- By specifying the internal name of the transfer request. This limits the search by FT$SUB to a single request.

- By specifying the external name of the transfer request. This limits the search by FT$SUB to requests with that external name.

- By specifying the null string. This places no limits on the search by FT$SUB.

The internal_name argument is an input/output argument. It controls the way the search for the transfer request specified in request_name is conducted. Before calling FT$SUB, the calling program sets internal_name to one of the following two character strings.

- The null string. In this case, FT$SUB will begin searching its data base from the beginning of the queue. Before the initial call to FT$SUB, internal_name must be the null string.

- The internal name of a transfer request. This is used only after an initial call to FT$SUB has taken place. This will cause FT$SUB to begin searching its data base with the request following the request having the specified internal name.

As an output argument, internal_name contains the internal name of the first transfer request meeting the requirements imposed by request_name, internal_name, and queue. However, if FT$SUB was unable to find such a request, the error code E$EOF will be returned in code, and the contents of internal_name will be null.

Therefore, the calling program can iteratively call FT$SUB over a set of file transfer requests, optionally limiting the set to include only requests with a specified external name. It does this by:

- Using the external name or the null string for request_name.

- Setting internal_name to the null string for the first call to FT$SUB, and using the returned internal_name argument on subsequent calls, until no more transfer requests are found.

The following table is an example of how this procedure might work in a typical environment. Each line of the table shows the values of code, internal_name, and request_name before a call to FT$SUB. An imaginary call to FT$SUB occurs between each line. Indeterminate or unimportant values are indicated by —.

Code	Internal Name	Request Name
—	''	'MEMO_1'
0	'12'	'MEMO_1'
0	'16'	'MEMO_1'
E$EOF	''	—

In the above example, two requests with the external name MEMO_1 were returned to the calling program. The first request had the internal name 12, and the second request had 16.

There are two further limitations that can be imposed on the set of transfer requests returned by FT$SUB. First, the requests can be limited to only those that belong to the calling user. This is normally the case. However, the restriction is lifted if the key parameter in the FT$SUB call is set to F$STAL, or if the calling user is SYSTEM.

Second, the set can be limited to only those requests in a certain FTS queue, by specifying a non-null queue argument.

Details follow on how to write a program that loops over the sets mentioned above.

Operating on All Requests on the System: To operate on all requests on the system, using the F$STAL key, for example, the calling program follows this basic form:

```
/* Initialize the request_data.valid bit to start up the do-loop,
   the found_request bit to indicate that no requests have been
   found, and initialize the internal name to null. */

request_data.valid='1'b;
found_request='0'b;
internal_name='';

/* Loop over the set of all FTS requests on the system.       */

do while(request_data.valid);  /* While there are requests.   */
   call ft$sub(f$stal,'',internal_name,'','','','',0,
               addr(request_data),1,null(),0,code);
   if code=0
      then do;                 /* If success, display the data.*/
           call display_request_data(request_data);
           found_request='1'b; /* Remember we found a request. */
           end;                /* if code=0 */
   end;                        /* do while(request_data.valid) */

if code=e$eof
   then if ^found_request then call tnou('No requests in system.',
                                         22);
   else if code=0 then call tnou('Program error, code=0.',22);
           else call fts_error(code);  /* Display error message.*/
```

Notice that the second argument, request_name, is a null string. This lifts the restriction that the returned requests all have a particular external name. When FT$SUB returns a request, the calling program can determine the external name of the request by examining request_data.extnam, as described below.

Notice also that the program flow ignores errors returned from FT$SUB as long as FT$SUB manages to return a valid request_data structure (as indicated by request_data.valid being set to '1'b). This is because the program is simply outputting the status of all the requests on the system, and is willing to ignore error conditions that do not interrupt its scan for requests. (See the sections entitled Error Codes and The Request Data Structure, below, for more information on the returned error code and its relationship to the request_data structure.)

Operating on All Requests for the Calling User: To perform an operation on all requests that belong to the user calling FT$SUB, and using the F$ABRT key, for example, the calling program should have the following form.

```
/* Initialize the request_data.valid bit to start up the do-loop,
   the found_request bit to indicate that no requests have been
   found, and initialize the internal name to null. */

request_data.valid='1'b;
found_request='0'b;
internal_name='';

/* Loop over the set of all FTS requests for this user.          */

do while(request_data.valid);  /* While there are requests.      */
   call ft$sub(f$abrt,'',internal_name,'','','','',0,
               addr(request_data),1,null(),0,code);
   if code=0
      then do;                 /* If success, display a message. */
           call display_abort_message(request_data);
           found_request='1'b; /* Remember we found a request.   */
           end;                /* if code=0                      */
      else if code^=e$eof
              then do;  /* Error, print the error code. */
                   call fts_error(code);  /* Do error message. */
                   if request_data.valid  /* If the request data is
                                             present, */
                      then call display_request_info(request_data);
                      /* Display summary information on request. */
                   end;  /* if code^=e$eof */
           end;          /* if code^=0                             */
   end;                  /* do while(request_data.valid)           */

if code=e$eof
   then if ^found_request then call tnou('No requests found.',18);
   else if code=0 then call tnou('Program error, code=0.',22);
```

Notice that the second argument, internal_name, is a null string. This lifts the restriction that the returned requests all have a particular external name. When FT$SUB returns a request, the calling program can determine the external name of the request by examining request_data.extnam, as described below.

Also notice how the error code returned from FT$SUB is handled. Information on the error code is always printed, but more descriptive information on the offending request is displayed only if the request information is present. In addition, the main do-loop continues as long as a valid request was found, even if the abort operation itself failed. This way, the entire FTS data base is scanned.

See the sections entitled Error Codes and The Request Data Structure, below, for more information on the returned error code and its relationship to the request_data structure.

Operating on All Requests With a Specific External Name: You can perform an operation on all requests with a particular external name that belong to the user calling FT$SUB. If you use the F$HOLD key, for example, the calling program should have the following form.

```
/* Initialize the request_data.valid bit to start up the do-loop,
   the found_request bit to indicate that no requests have been
   found, and initialize the internal name to null. */

request_data.valid='1'b;
found_request='0'b;
internal_name='';
request_name='MEMO_TO_MARK';

/* Loop over the set of all FTS requests for this user.          */

do while(request_data.valid); /* While there are requests.       */
   call ft$sub(f$hold,request_name,internal_name,'','','','','1'b,
               addr(request_data),1,null(),0,code);
   if code=0
      then do;                    /* If success, display a message. */
           call display_hold_message(request_data);
           found_request='1'b; /* Remember we found a request.    */
           end;                   /* if code=0                      */
      else if code^=e$eof
              then do;  /* Error, print the error code. */
                   call fts_error(code);  /* Do error message. */
                   if request_data.valid  /* If the request data is
                                             present, */
                      then call display_request_info(request_data);
                      /* Display summary information on request. */
                   end;  /* if code^=e$eof */
           end;          /* if code^=0                             */
   end;                  /* do while(request_data.valid)           */

if code=e$eof
   then if ^found_request then call tnou('No requests found.',18);
   else if code=0 then call tnou('Program error, code=0.',22);
```

The second argument, request_name, contains the external name specified by the user. This imposes the restriction that the returned requests all have the external name request_name.

Also notice how the error code returned from FT$SUB is handled. Information on the error code is always printed, but more descriptive information on the offending request is displayed only if the requested information is present. In addition, the main do-loop continues as long as a valid request was found, even if the hold operation itself failed. This way, the entire FTS data base is scanned.

See the sections entitled Error Codes and The Request Data Structure, below, for more information on the returned error code and its relationship to the request_data structure.

Error Codes

After a call to FT$SUB, the status of the call is returned in code. If the returned value is nonzero, the requested operation was not performed. However, some of the side effects of the call may have been performed, depending on the actual value set in code.

For example, suppose a call is made to put a transfer request on hold, and the specified request is already on hold. The returned internal_name argument and the request_data structures will contain the appropriate data for the specified transfer request, even though the status of the request is not changed. This allows the calling program to determine the present status of the specified request, and to continue scanning for other requests.

In all cases, a non-zero error code is accompanied by valid information in error_data. See the section below, entitled The Error Data Structure for more information.

The list below contains error codes that may be returned by FT$SUB during normal operation. Other error codes may be returned, but these usually indicate an unusual circumstance, such as a disk error, or insufficient disk storage to submit a request. Error codes marked with * indicate operations that failed but still return valid internal_name and request_data values.

Code	Meaning
E$BKEY	Bad key in call.
E$EOF	End of file.
E$BPAR	Bad parameter.
F$ARTL	Argument too long.
F$INEX	Invalid external name.
F$NERF	No eligible request of this name found.

	F$NRFD	No request of this name found.
	F$OMOP	Only one management option allowed.
*	F$TRPR	Transfer in progress.
*	F$RQHU	Request already put on hold by user.
*	F$RQHO	Request already put on hold by operator.
*	F$RQHF	Request already put on hold by FTS.
*	F$RQAB	Request already aborted.
*	F$RQWT	Request waiting.
*	F$RHPR	Request held by operator.
	F$QNMD	Queue name may not be modified.
	F$NCMD	NO_COPY flag may not be modified.
	F$CPMD	COPY flag may not be modified.
	F$SSMD	Source site may not be modified.
	F$DSMD	Destination site may not be modified.
	F$HDMD	HOLD flag may not be modified.
	F$SFMD	Source file type may not be modified.
	F$DFMD	Destination file type may not be modified.
	F$CLMN	Command lines must be null.
	F$NWNA	Networks unavailable.
	Q$NVDB	The FTS data base is invalid.
	Q$QNRD	FTS not ready to use.

The Request Data Structure

When the request_data structure is provided during a call to FT$SUB, it is filled in by FT$SUB to indicate the status and parameters of the specified request. However, it is filled in only if the returned error code is non-zero, or if it is one of the values described in the section above.

To simplify the process of determining the validity of request_data, a valid bit is provided in the substructure request_data.flags. If this

bit is '1'b, the entire request_data structure contains valid information. Otherwise, the contents of request_data are not valid.

Note, however, that if the request_data structure was returned from FT$SUB as a result of a call using the F$STAL key (Status of all requests), certain fields in request_data are set to null or zero values by FT$SUB before returning to the caller to prevent the FT$SUB caller getting access to other user's confidential information. These fields are:

```
log_file
site(source_ptr)
site(destination_ptr)
user_pswd(source_ptr)
user_pswd(destination_ptr)
tree(source_ptr)
tree(destination_ptr)
file_pswd(source_ptr)
file_pswd(destination_ptr)
account_pswd(source_ptr)
account_pswd(destination_ptr)
device
```

The source_ptr and destination_ptr indexes are constant expressions that you might find useful in your program. They are used to access one of the two elements in several different arrays in request_data in a mnemonic fashion. Use the following statement in your PL1G program to set up source_ptr and destination_ptr correctly.

```
%replace source_ptr by 1,
        destination_ptr by 2;
```

The request_data structure and its corresponding version number are designed to allow future changes to the structure contents, while supporting programs that call FT$SUB by using an earlier version of the structure. This structure has the following declaration, known as version 1.

```
dcl 1 request_data based,       /* Beginning of item entry.        */
      2 inumber bin,            /* Internal use only.              */
      2 iorig bin,              /* Internal use only.              */
      2 istatus bin,            /* Status of item, see below.      */
      2 reserved bin(31),       /* Reserved.       */
      2 userid char(32) var,    /* User name.                      */
      2 frnode char(32) var,    /* Reserved.    */
      2 extnam char(32) var,    /* External name for this request.*/
      2 tempfile char(32) var,  /* Temporary file and internal name
                                   for this request. */
      2 idate char(8),  /* Date this request queued (YYYYMMDD).*/
      2 itime bin(31),  /* Time this request queued (seconds after
                           midnight). */
      2 netctl bin,     /* Internal use only.                      */
      2 version bin,    /* Version number of request block (1). */
```

```
      2 block_type bin, /* Type of request block.
                           0 = transfer request block. */
      2 flags,          /* General flag halfword.          */
        3 valid bit(1), /* True, '1'b, if request block
                           contains valid info, otherwise
                           set to false, '0'b.             */
        3 mbz bit(15),  /* Pad flags to one halfword.      */
      2 last_date char(8), /* Date of last connect (YYYYMMDD).
                              Valid only if ntry>0.        */
      2 last_time bin(31), /* Time of last connect (minutes). */
                              Valid only if ntry>0.        */
      2 ntry bin,       /* Number of connection attempts.  */
      2 action bin,     /* Transfer action code, see below. */
      2 priority bin,   /* Reserved.      */
      2 file_size bin(31), /* File size (bytes).           */
      2 defer,          /* Reserved. */
        3 time bin,
        3 date char(8),
      2 runby,          /* Reserved. */
        3 time bin,
        3 date char(8),
      2 runevery bin(31), /* Reserved. */
      2 pad bit(5),     /* Pad up to 16 bits.              */
      2 append bit(1),  /* Reserved.          */
      2 file_exist bit(2), /* Required existence of output file. */
      2 file_type bit(1), /* Binary = '1'b, text = '0'b.   */
      2 copy bit(1),    /* Make copy of file = '1'b.       */
      2 delete bit(1),  /* Delete local source file after  */
                        /* transfer = '1'b. */
      2 defer_set bit(1), /* Reserved. */
      2 runby_set bit(1), /* Reserved. */
      2 runevery_set bit(1),/* Reserved. */
      2 notify(2) bit(1), /* Source/dest user notify on = '1'b. */
      2 log_file char(128) var, /* User requested log file. */
                           Null if no file specified.  */
      2 msg_level bin,  /* Message level for logging (1-4). */
      2 site(2) char(128) var, /* Source/dest site addresses. */
      2 site_type(2) bin,   /* Site is Prime = 1, Other = 0. */
      2 stream char(32) var, /* Name of request queue.     */
      2 user(2) char(32) var, /* Source/dest users.        */
      2 user_pswd(2) char(32) var, /* Reserved. */
      2 tree(2) char(128) var,  /* Source/dest file names. */
      2 file_pswd(2) char(32) var, /* Source/dest file passwords.*/
      2 kinship(2) char(32) var, /* Source/dest file types.
                                    SAM, DAM, SEGSAM, SEGDAM. */
      2 mode(2) bin,    /* Reserved. */
      2 account(2) char(32) var,   /* Reserved. */
      2 account_pswd(2) char(32) var, /* Reserved. */
      2 device char(32) var,  /* Destination device, or null. */
      2 tid bin,        /* Reserved. */
      2 response_queue char(32) var, /* Reserved. */
      2 option char(255) var; /* Reserved. */

%replace request_size by 825; /* Request_data size in halfwords   */
```

Status of Item: The value of istatus is set according to the following table.

```
/* Request status values, - permitted values of request.istatus. */

%replace wait_rqstatus by 1;  /* Waiting. */
%replace busy_rqstatus by 2;  /* In progress (transferring). */
%replace user_hold_rqstatus by 3;  /* Held by user. */
%replace operator_hold_rqstatus by 4;  /* Held by FTS operator
                                          (user SYSTEM). */
%replace user_abort_rqstatus by 5;  /* Aborted by user. */
%replace fts_hold_rqstatus by 6;  /* Held by FTS server. */
%replace operator_abort_rqstatus by 7;  /* Aborted by FTS operator
                                           (user SYSTEM). */
```

File Transfer Action Code: The value of action is set according to the following table.

```
/* Definition of the possible values for request.action */

%replace null_action by 0;  /* Initial value. */
%replace take_file_action by 1;  /* File is being sent. */
%replace give_file_action by 2;  /* File is being fetched. */
%replace take_jobin_action by 3;  /* Not used. */
%replace give_jobin_action by 4;  /* Not used. */
%replace take_jobout_action by 5;  /* File is being sent to a
                                      device. */
%replace give_jobout_action by 6;  /* Not used. */
```

The Error Data Structure

If a call to FT$SUB results in code being set to a non-zero value, then you can peruse the error_data structure, if it is passed to FT$SUB, to pinpoint the offending input. This is useful if the error code indicates an error in one of the two command lines passed to FT$SUB, user_cmdl and prog_cmdl. Therefore, this structure is useful only when FT$SUB is being used to submit or modify file transfer requests.

The following table shows error codes that indicate an error in user_cmdl or prog_cmdl.

Code	Meaning
F$ARTL	Argument too long.
F$BDCL	Bad command line format.
F$BDDN	Bad device name.
F$BDKW	Unknown keyword.

F$BDSN Bad site name format.

F$CNOP Conflicting option.

F$CPLS Copy option only applies to local source file.

F$CPMD Copy flag may not be modified.

F$DFMD Destination file may not be modified.

F$DFNS Destination file has not been specified.

F$DLLS Delete option only applies to local source file.

F$DRNA Device transfer from remote site not allowed.

F$DSMD Destination site may not be modified.

F$DSNC Destination site is not configured.

F$DUIN Destination user name invalid.

F$DUNS Destination user not specified when destination notify requested.

F$DUOP Duplicate option.

F$FPTL Full pathname too long.

F$HDMD Hold flag may not be modified.

F$IDFT Invalid destination file type.

F$IFDC Illegal file or directory conversion.

F$INMS Invalid message level.

F$ISFT Invalid source file type.

F$MCLP Missing command line parameter.

F$MBNL Message level specified but request log treename omitted.

F$NCLS No copy option only applies to local source file.

F$NCMD No Copy flag may not be modified.

F$NDLS No delete option applies only to local source file.

F$NTCF Not configured.

F$OMOP Only one management option allowed.

F$PINS	Segment dir. transfer to/from a Rev 1 site is not supported.
F$PSFQ	Passworded pathname must be fully qualified.
F$RTIS	Remote treename incorrectly specified.
F$QNMD	Queue name may not be modified.
F$RLST	Request log treename same as source or target treename !!
F$SDSL	Source or destination site must be local.
F$SFMD	Source file type may not be modified.
F$SFNE	Source file does not exist.
F$SFNS	Source file nas not been specified.
F$SFTD	Specified and actual source file types differ.
F$SSMD	Source site may not be modified.
F$SSNC	Source site is not configured.
F$SUIN	Source user name invalid.
F$SUNS	Source user not specified when source notify requested.
F$TDFN	Transfer to a device as well as a file is not allowed.
F$TDNS	Transferring a SEG directory to a DEVICE is not supported.
F$TFNP	Transferring a file to itself is not possible.
F$UNOP	Unknown option.

However, to simplify programming, the error_data structure itself has a bit that indicates the validity of its data. This is the valid bit. It is set to '0'b by FT$SUB whenever FT$SUB is unable to fill the rest of the structure with valid data. Otherwise, valid is set to '1'b, and the rest of the structure contains valid data.

Before calling FT$SUB, your program should initialize start_ptr and end_ptr to 0. FT$SUB adds to these values the starting and ending locations in the command line of the offending option or keyword. After FT$SUB returns a non-zero error code, if comm_line in the error_data structure is non-zero, display the specified command line to the user, and use the start_ptr and end_ptr values to indicate which part of the command line was in error. If these values were initialized to 0, then they will contain values between 1 and the length of the offending command line, inclusive.

The error_data structure and its corresponding version number are designed to allow future changes to the structure contents, while supporting programs that call FT$SUB by using an earlier version of the structure. This structure has the following declaration, known as version 1.

```
dcl 1 error_data based,
      2 valid bit(1),  /* '1'b if structure info valid, else '0'b. */
      2 mbz bit(15),   /* Bit padding. */
      2 version bin,   /* Version number of buffer, must be 1.     */
      2 errcode bin,   /* Copy of FT$SUB return code.              */
      2 comm_line bin, /* 1 = Display program command line with ptrs.
                          2 = Display user command line with ptrs.
                          0 = Don't display any command line.     */
      2 start_ptr bin, /* If comm_line ^= 0, points to start of
                          area on command line causing the error. */
      2 end_ptr bin,   /* If comm_line ^= 0, points to end of
                        area on command line causing the error.
                        FT$SUB adds to the value supplied to
                        it in start_ptr and end_ptr.      */
      2 text char(160) var, /* Optional explanatory text.         */
      2 proc char(32) var;  /* Name of procedure detecting
                               the error.                         */

%replace error_size by 104;  /* Size of error_data in
                                  16-bit halfwords.  */
```

You can use the text and proc strings during a call to the system error printing subroutine ERRPR$. They provide additional visual feedback on the error, even if the problem was not in one of the command lines.

EXAMPLE

Here is a sample program using FT$SUB.

```
C ft$sub_test.f77. Submit a local file to FTS.
C
C This program reads the source filename, destination filename, and
C destination site from the user terminal, and then submits this
C request by calling FT$SUB.
C
      PROGRAM main
C
$INSERT SYSCOM>FT$SUB.INS.FTN
C
      INTEGER*2 empty_string(17),   /* for char*32 var
     +          empty_string1(129), /* for char*255 var
     +          options(21)         /* for char*40 var
C
```

```
      INTEGER*2 int_name(17), int_length  /* for char*32 var
      CHARACTER*32 int_string
      EQUIVALENCE (int_length,int_name(1)), (int_string,int_name(2))
C
      INTEGER*2 cmd_line(41), cmd_length  /* for char*80 var
      CHARACTER*80 cmd_string
      EQUIVALENCE (cmd_length,cmd_line(1)), (cmd_string,cmd_line(2))
C
C Define request block storage.
C
      INTEGER*2 request_block(900)
C
C Define error block storage and structure.
C
      INTEGER*2 error_data(104),    /* Total size
     +      e_valid,
     +      e_text_len,
     +      e_proc_len
      CHARACTER*160 e_text
      CHARACTER*32 e_proc
      EQUIVALENCE (e_valid, error_data(1)),
     +      (e_text_len, error_data(7)),
     +      (e_text, error_data(8)),
     +      (e_proc_len, error_data(88)),
     +      (e_proc, error_data(89))
C
      INTEGER*2 return_code
C
      COMMON /ALAN/empty_string,
     +      empty_stringl,
     +      options,
     +      int_name,
     +      cmd_line,
     +      request_block,
     +      error_data
C
C Initialize several FT$SUB parameter strings.
C
      CALL init_str(empty_string, empty_stringl, int_name, options)
C
C Now get the user command line
C
      CALL get_line(cmd_string, cmd_length)
C
C Dispatch the request!
C
      CALL FT$SUB(F$SUBM,
     +      empty_string,
     +      int_name,
     +      cmd_line,
     +      options,
     +      empty_stringl,
     +      empty_string,
     +      F$NQRY,
```

```
     +      LOC(request_block), 1,
     +      LOC(error_data), 1,
     +      return_code)
C
C If the submission was OK, type out the internal name, otherwise
C dump error message.
C
      CALL TNOUA('Submitted as ', INTS(13))
      CALL TNOU(int_string, int_length)
C
      CALL TNOUA('Submission error code: ', INTS(23))
      CALL TODEC(return_code)
      CALL TONL
      IF (AND(e_valid,:100000) .NE. 0) THEN
         CALL TNOUA(e_text, e_text_len)
         CALL TNOUA(' (', INTS(2))
         CALL TNOUA(e_proc, e_proc_len)
         CALL TNOU(') ', INTS(1))
      ENDIF
C
      CALL EXIT
      END
C GET_LINE
C
      SUBROUTINE get_line(string, length)
C
      CHARACTER*80 string
      INTEGER*2 length
C
C Get the source file, the destination file, and the destination site
C
      PRINT 9000
      READ (1,'(A80)') string
      length = 80
C
9000  FORMAT ('Give srcfile, destfile, -ds destsite')
      RETURN
      END
C INIT_STR
C
C This routine initializes several FT$SUB parameter strings.
C
      SUBROUTINE init_str(e, el, int, o)
C
      INTEGER*2 e(1), el(1), int(1), o(1)
      INTEGER*2 defopt(20)          /* Max 40 chars
C
      DATA defopt/'-log demo.log                              '/
C
      e(1) = 0                  /* Set null length
      el(1) = 0
      int(1) = 0
.C
      o(1) = 13                 /* Actually used
```

```
      DO 10 i = 1, 20
10    o(i + 1) = defopt(i)
C
      RETURN
      END
```

PART VII

Operator's Guide to PRIMENET

18

Monitoring PRIMENET and FTS

INTRODUCTION

PART VII of the PRIMENET Guide describes the system operator's responsibilities to networking software. The following duties are described briefly in this chapter and fully in the chapters listed below.

- Monitoring the PRIMENET* directory — in this chapter
- Adding remote disks -- in this chapter
- Starting and stopping PRIMENET — Chapter 19
- Monitoring network servers — Chapter 20
- Monitoring network events — Chapter 21
- Monitoring FTS — Chapter 22
- Monitoring RINGNET — Chapter 23

This chapter contains information on files that are used in network processing, as well as access rights those processes must have to use those files in the PRIMENET* directory.

MONITORING THE PRIMENET* DIRECTORY

PRIMENET* is a top-level directory that must exist on the system disk (defined by the COMDEV directive in the system configuration file) at system startup. This directory contains all files needed to run PRIMENET, including network event-logging files and Remote File Access, whose slaves use this directory.

As an operator, you are responsible for checking the PRIMENET* directory periodically. To do so, use the ATTACH and LD commands to list the directories on the supervisor terminal, or use the COMOUTPUT command to write the directory listings to a file.

The following two sections describe what files exist in the PRIMENET* directory, and what access rights should be provided for the various network processes that use those files.

PRIMENET* Directory Files

The PRIMENET* directory must contain the following files and be accessible to network servers in order for them to run properly.

- The network log file called NET_LOG.mm/dd/yy. This file is generated by PRIMOS when network logging is turned on. The PRINT_NETLOG and LOGPRT -NET commands use this file to generate an output log file in the current directory called NETLST. The PRIMENET* directory must be present for logging to take place.
- A command file called SLAVE.COMI that initiates the startup of slaves on the system.
- The Route-through run-time command file.
- The NETWORK_SERVER.COMI and NETMAN.SAVE files, which are run at system startup as part of the initialization of NETMAN, the network server process.

The following is a sample of the contents of a typical PRIMENET* directory.

```
OK, LD

<BRANCH>PRIMENET* (DALURW access)
23 records in this directory, 23 total records out of quota of 10000.

4 Files.

NETMAN.SAVE        NETWORK_SERVER.COMI              NET_LOG.01/17/84
SLAVE.COMI

.OK,
```

Access Rights

NETMAN must have ALL access to the PRIMENET* directory, or the network is not set up. In this case, NETMAN logs out, and the message "Network Server logged out during network startup" is displayed at the supervisor terminal. NETMAN also needs at least ALURW rights to the PRIMENET* directory to write network events to the NETLOG log file.

RT_SERVER must have LUR rights to the PRIMENET* directory to access files that control its activity.

The user-id SYSTEM should have ALL rights to the PRIMENET* directory, since it is from this user-id that you control the network server process. See the System Operator's Guide for more information on using ACLs to assign access rights.

System Administrators and operators need LURWD rights in order to purge or write to the network log file as well. All other users ($REST) should have LUR rights to this directory.

ADDING REMOTE DISKS

The ADDISK command allows users to access

- Disks on the current system
- Disks on remote systems (using the -ON option)

Usually, the ADDISK command is part of the system startup file. Any ADDISK commands must follow the START_NET command because ADDISK verifies that the remote node is configured before allowing you to add disks for that remote system. The System Operator's Guide, Part II describes the ADDISK command.

NETWORK CONFIGURATION FILE

The network configuration file, which is generated by the global configurator (CONFIG_NET), is generally kept in the PRIMENET* directory. The name of the configuration file is PRIMENET.CONFIG.

The System Administrator should have read and write access rights to the configuration file, since it is the Administrator that will be updating the file when the network changes.

STARTING AND STOPPING PRIMENET

The following commands start and stop PRIMENET on a node without disrupting local PRIMOS operation.

- START_NET
- STOP_NET

START_NET also starts the Route-through server.

MONITORING NETWORK SERVERS

Network servers are processes that ensure that PRIMENET is up and running, allowing systems in a network to interact with one another. The operator must monitor the following communications servers with the STATUS NETWORK and STATUS USERS commands.

- NETMAN
- RT_SERVER

NETMAN is the PRIMENET server process that allows systems in a network to communicate with one another. RT_SERVER is the Route-through server that controls the virtual circuit gateways between indirectly connected nodes in a network. (These indirect or next-hop nodes are configurable with CONFIG_NET.)

MONITORING NETWORK EVENTS

Network events are listed in a file that lists various occurrences between systems on a network. The following commands can be used to monitor network events.

- PRINT_NETLOG
- LOGPRT -NET

They both generate the same output file, called NETLST. In order for them to create this file, which lists network events over a period of time, network event logging must be turned on through the NETREC CONFIG directive.

MONITORING FTS

FTS is the File Transfer Service, which includes a file transfer utility (FTR) that can be used between Prime systems. This service must be configured on each system on which it is being used. Operators must control and monitor FTS through the following actions.

- Start the FTS Manager (YTSMAN).
- Start the FTS server.
- Make sure user requests are sent as swiftly as possible.
- Archive FTS log files from the FTSQ* directory.

MONITORING RINGNET

RINGNET is Prime's Local Area Network that connects Prime computers through a PRIMENET Node Controller (PNC) and controls data throughput. The following ring monitor programs gauge the system's accessibility and activity on a ring network.

- MONITOR_RING
- FIND_RING_BREAK

MONITOR_RING displays network throughput and error statistics. It does not monitor the signals on the ring itself; it monitors ring traffic on the node that it is run on.

FIND_RING_BREAK shows you the location of breaks in the ring. This program works only for hardware breaks that cause complete signal interruption on the ring. FIND_RING_BREAK can be useful when you use it in conjunction with MONITOR_RING to determine a break.

19

Starting and Stopping PRIMENET

INTRODUCTION

The following commands allow an operator to add or remove a system from a PRIMENET network without interrupting local PRIMOS system activity.

- START_NET
- STOP_NET

Both of these commands are external PRIMOS commands; you invoke START_NET to bring up PRIMENET on a system and STOP_NET to shut down PRIMENET on a system. You can only use these commands when you are logged in as the user-id SYSTEM or from the supervisor's terminal. Furthermore, START_NET must have READ access to the network configuration file.

This chapter describes

- NET ON versus START_NET
- What each command does
- Command line format for each command

Error messages for START_NET and STOP_NET are described in Appendix D.

NET ON VERSUS START_NET

Prior to Rev. 19.3, the NET ON directive was used to start PRIMENET. The START_NET command replaces this directive. The NET ON directive can remain in the system configuration file, since it does not affect network operation.

THE START_NET COMMAND

Use START_NET to start PRIMENET on the system. START_NET determines from the information in the global network configuration file where the system is located in the network. The Route-through server is also activated if the system has been configured as a gateway node to indirectly-connected systems.

You can issue START_NET at any time. Normally, it is part of system startup. You should place START_NET at the beginning of the system configuration file to allow the network enough time to initialize. START_NET must precede any ADDISK commands because ADDISK checks to see if a remote node is configured before letting you add disks for that system. However, you can place the START_NET command either before or after shared libraries.

INVOKING START_NET

The syntax of START_NET is:

```
START_NET -NODE nodename [config_pathname]   -TRACING_NODE
                                             -HELP
```

-NODE	Specifies the name of the local node. An incorrect node name causes conflicts to occur when START_NET reads the network configuration file.
config_pathname	The pathname of the network configuration file. If no pathname is given, the default pathname, PRIMENET*>PRIMENET.CONFIG, is used.
-HELP	Describes the complete syntax and options of the START_NET command.
-TRACING_NODE	Enables the trace option when the MONITOR_RING program is used. See Chapter 23 for more information on this program.

When START_NET is invoked, an acknowledgement that the network initialization is beginning and any indirect connections are displayed. In the example below, pathname is the pathname of the configuration file (usually PRIMENET*>PRIMENET.CONFIG), n is the revision number of the file noted in pathname (n is incremented each time the file is saved), nodename is the system on which START_NET is being run, and indirect connection is the path between nodename and destination node.

```
Beginning Network Initialization
File: pathname Rev n
(Config_Net rev. 19.3)

*** Node: nodename ***
Indirect connection:  <nodename>next-hop node>destination node
```

THE STOP_NET COMMAND

Use the STOP_NET command to shut down PRIMENET on the system and remove it from the network. Invoking STOP_NET does not affect local activity. STOP_NET does, however, disable network processes on the system. If FTS is running, the System Administrator must perform the following tasks before shutting down the network.

- Stop the file servers with the FTOP -STOP_SRVR command.
- Stop the file manager (YTSMAN) with the FTOP -STOP_MNGR command.
- Issue the STOP_NET command.

Note

Before restarting the network, use the FTOP START_SRVR and FTOP START_MNGR commands to restart the FTS servers and manager.

The list below describes what happens on a node on which the STOP_NET command is issued.

- Remote disks are shut down.
- Open virtual circuits are cleared.
- NPX slaves go to sleep.
- The Route-through server logs out.
- NETMAN logs out.

When a system is disconnecting from the network, remote users or applications attempting to access it are notified. Messages that appear for each type of remote access are listed below.

Remote file access users see the following message.

```
DISK HAS BEEN SHUT DOWN.
```

Remote users (logged in with the LOGIN -ON command) see the following message.

```
HOST DOWN.
```

NETLINK users see the following message.

```
nodename DISCONNECTED (NETWORK SERVER LOGGED OUT).
```

The following events occur with IPCF users (applications). An application with a virtual circuit established to the node on which STOP_NET is invoked sees the circuit cleared, and the diagnostic code CD$NSV returned. If the application has not established a virtual circuit and is waiting for an incoming call, the diagnostic code XS$NET is returned.

INVOKING STOP_NET

The syntax for STOP_NET is:

```
STOP_NET [-HELP]
```

The -HELP option describes the complete syntax of the STOP_NET command.

20

Monitoring Network Servers

INTRODUCTION

As an operator, you monitor PRIMOS in various ways. Some of these methods are described in the System Operator's Guide. This chapter describes how to monitor network servers. It contains information about.

- The PRIMENET server, NETMAN.
- The Route-through server, RT_SERVER.
- The STATUS USERS command.
- The STATUS NETWORK command.

NETMAN and RT_SERVER are phantom processes that are activated to handle their corresponding functions, PRIMENET and Route-through. These functions run from the PRIMENET* directory, which is described in Chapter 18.

MONITORING THE NETWORK SERVER PROCESS (NETMAN)

The network server process is called NETMAN. NETMAN appears on the STATUS USERS list. An example of STATUS USERS is provided later in this chapter.

NETMAN does not have to be registered in the user validation file. However, NETMAN comes from the system pool of phantom processes, so your system must have at least one phantom configured exclusively for NETMAN's use. (Refer to the Network Planning and Administration Guide for information on CONFIG directives related to networking.)

MONITORING THE ROUTE-THROUGH SERVER (RT_SERVER)

The Route-through server is a process that allows a system to act as a "gateway" for communication between nodes that are not directly connected or not on a ring. The Route-through server may be monitored with the STATUS NETWORK command.

You should assign the Route-through server, RT_SERVER, LUR access rights to the PRIMENET* directory.

Usually, the START_NET command starts the Route-through server, but in some instances, the command line must be typed by the system operator at the supervisor terminal.

USING STATUS USERS

The STATUS USERS command lists all active processes, including the network server processes, NETMAN and RT_SERVER. The following example shows a STATUS USERS command. The network servers have a line-id of "nsp" (network server process) and "rts" (route-through server), respectively.

```
OK, STATUS USERS

User                    No  Line Devices
SYSTEM                   1   asr <SYSDSK> AL077
STANDISH                 2     0 <BRANCH>
ANTHONY                  3     1 <BRANCH>
JONATHAN                 4     2 <BRANCH> <LEAF>
MARY                     6     4 <BRANCH> <LEAF>
NETMAN                  85   nsp <BRANCH>          <—
RT_SERVER               92   rts <BRANCH>          <—
BATCH_SERVICE           94 phant <BRANCH> (2)
BACKUP_SERVICE          95 phant <BRANCH> <LEAF> (0)
YTSMAN                  98 phant <BRANCH>
FTP                     99 phant <BRANCH>

OK,
```

USING STATUS NETWORK

The STATUS NETWORK command lists any system on which NETMAN and the Route-through server are running. The following example shows a STATUS NETWORK command on the system OAK. This command also shows the Route-through nodes, which PRIMENET uses to connect indirect nodes.

```
OK, STATUS NETWORK
OAK stat net


Full duplex network


      Node        State
    OAK           ****
    TELENET        Up

Ring network


      Node        State
    OAK           ****
    LINDEN         Up
    ASH            Up      /* A local RING network */
    PINE           Up
    MAPLE          Up
    FIR           Down

Public data network

      Node
    EUROPA                 /* A PDN connected to the local network */

Route-through network

      Node
    VINE                   /* These are the gateway nodes */
    BRANCH
    LIMB
    LADDER
```

21

Monitoring Network Events

INTRODUCTION

PRIMOS provides you with an event logging mechanism that records information about significant network events. The NETREC CONFIG directive or the EVENT_LOG -NET command turn on the mechanism. Refer to the Network Planning and Administration Guide for more information on network-related configuration directives. The EVENT_LOG command is described in the System Operator's Guide.

This chapter describes the following.

- The PRINT_NETLOG command
- Network event types
- Controlling the size of the NETLOG file
- Network event messages
- PRINT_NETLOG error messages

You can also specify new network event types with PRINT_NETLOG. See the System Operator's Guide for more information on this.

THE PRINT_NETLOG COMMAND

The PRINT_NETLOG command is a newer version of LOGPRT, but only LOGPRT can run under PRIMOS II. LOGPRT is described in the System Operator's Guide. Both the PRINT_NETLOG and the LOGPRT commands generate output from the NET_LOG file in a NETLST file. The following shows the command format of the PRINT_NETLOG command.

```
PRINT_NETLOG { [output-file] } [options]
             { TTY           }
```

The default output file is NETLST. You can specify another filename. If you specify TTY, output will be displayed on your terminal.

Note

A file unit is opened whenever event logging is turned on. This file unit can be closed only if you turn event-logging off. (The CLOSE command doesn't close it.)

You can specify any NETLOG file with the -INPUT option, which is described below. If you do not include the -INPUT option on the PRINT_NETLOG command line, PRINT_NETLOG uses the most recently created NETLOG file in the PRIMENET* directory. If PRINT_NETLOG is unable to find a network event log file, it prompts for an input file name. The following options are available.

Option	Description
-HELP	Print a list of PRINT_NETLOG options. The PRINT_NETLOG command must be retyped after the options are printed.
-INPUT pathname	Specifies the pathname of the input log file to be processed. If this option is not present on the command line, PRINT_NETLOG will attempt to use the most recently created network event log file, as described above.
-FROM mmddyy [hhmm] TODAY	Only entries from the specified date to the latest entry are processed. Specify TODAY instead of mmddyy to refer to today's date. Following the date specification, an optional time specification of the form hhmm (hours, minutes) may be entered. A time entry may be between 0000 and 2359. Omitting the time specification is equivalent to specifying '0000'. PRINT_NETLOG checks each entry individually to see if its date/time stamp indicates

that it should be formatted. An out-of-sequence entry (for example, the wrong date entered by the operator) will not turn on entry formatting prematurely.

-TYPE type type ... Process entries only of the indicated types. Network types are described in the next section.

The time and date stamps associated with the selected entries will not be processed unless TIMDAT is explicitly selected, for example, if you type -TYPE diskname TIMDAT, all disk errors and their associated time and date stamps will be processed. If TIMDAT alone is specified, all time/date stamps will be processed. If TIMDAT is specified in conjunction with one or more other types, only the time/dates of the selected types will be processed. If the -TYPE option is not specified, all entries will be processed.

-SPOOL Spool the output file when done. PRINT_NETLOG will print the name of the output spool file.

-DELETE Delete the output file when done. This option should be specified only when the -SPOOL option is also specified.

-PURGE Empty, but do not delete, the event log input file when event log processing is complete. Write access is required on the input file.

-CENSUS PRINT_NETLOG totals the entries for each event in the input file and writes the totals to the output file or terminal. Only non-zero totals are displayed.

-CONTINUE Continue after a bad entry is found. PRINT_NETLOG will normally halt if an invalid entry is encountered. If this option is specified, PRINT_NETLOG will continue processing in an attempt to find the next valid entry.

-DEBUG This option causes PRINT_NETLOG to read entries from the terminal and can be used for testing PRINT_NETLOG's formatting for entry types. Each entry should be entered as a series of tokens (using RDTK$$'s rules). Octal tokens are converted to

binary; all others are taken as ASCII strings. PRINT_NETLOG leaves this mode of operation whenever a 'Q', 'q', or null line is entered.

-REMARK text — Enter an event of type REMARK directly into the input file. This can be used by an operator who wishes to record an observation on some event that might affect the subsequent operation of the network. All text after the -REMARK option is taken as the text to be entered into the input file. Consequently, the -REMARK option must be the last option specified on the command line. The message can be up to 80 characters in length and need not be surrounded by apostrophes. Write access is required on the input file.

-DUMP — In addition to its normal formatting, PRINT_NETLOG will dump each entry processed in octal. This option is provided as an additional aid to those who define their own event types. Only those entries that have been selected for processing are dumped.

To display the contents of NET_LOG.mm/dd/yy, issue the PRINT_NETLOG command. The following example shows how a sample NETLST log might appear:

```
OK, PRINT_NETLOG TTY
PRINT_NETLOG REV 19.3


PRINT_NETLOG EVENT LOG FOR INPUT FILE <0>PRIMENET*>NET_LOG.11/10/83
14:31:32 THURSDAY NOVEMBER 10, 1983

00:00:08 THURSDAY NOVEMBER 10, 1983
--------------------------------

COLD START - PRIMOS REV REV.19.3

RING NODE:  137 NOT ACCEPTING XMITS.
NODE NOT IN RING

LEVEL III PROTOCOL DOWN - RING NODE:  137

RING NODE:  34 NOT ACCEPTING XMITS.
NODE NOT IN RING

LEVEL III PROTOCOL DOWN - RING NODE:  34
```

```
00:00:40 THURSDAY NOVEMBER 10, 1983
-----------------------------------

RING NODE:  39 NOT ACCEPTING XMITS.
NODE NOT IN RING

LEVEL III PROTOCOL DOWN - RING NODE:  39

00:10:40 THURSDAY NOVEMBER 10, 1983
-----------------------------------

NPX>IRNRCV MASTER'S CIRCUIT WAS CLEARED - NODE:  001003 (OCT)
VIRTUAL CIRCUIT STATE (1):  000004 (OCT), VIRTUAL CIRCUIT STATE (2)
:
000372 (OCT) (EVENT OCCURRED 4 TIMES)

TYPE    NUMBER
------  ------
COLD    2
TIMDAT  43
RESET   2
BADESQ  1
RING3   39
HOSTDN  42
NPXCLR  27
NPXCON  1

OK,
```

If you want to send the output to a printer instead of displaying it on your terminal, issue the following command:

```
PRINT_NETLOG -SPOOL -DELETE
```

The output file will be created, spooled, and then deleted. To choose an input file other than the most recent event log file, include the -INPUT option followed by the pathname of the input file on the command line, as follows:

```
PRINT_NETLOG TTY -INPUT PRIMENET*>NET_LOG.11/20/83
```

Network event log files are located in the directory PRIMENET*.

NETWORK EVENT TYPES

The following network types can be specified in the PRINT_NETLOG -TYPE command.

BADENT	Bad entry — not an entry
BADSEQ	Packets out of sequence
COLD	Cold starts
DIAPKT	A diagnostic packet was received from a PDN
HOSTDN	Level III protocols down
ICS1.0 (X.25)	Line not defined
ICS1.1 (X.25)	Deconfig code word not queued
ICS1.2 (X.25)	Logical connection deleted
ICS1.3 (X.25)	Could not delete logical connect
ICS1.4 (X.25)	LCADl not found in LCB
ICS1.5 (X.25)	Logical connection lost
ICS1.6 (X.25)	Flush timeout occurred
ICS1.7 (X.25)	Illegal flush complete
ICS1.8 (X.25)	Line not assigned
ICS1.9 (X.25)	Unspecified error
INCREQ	Incoming call request
LPE	Local procedure errors
NETDMP	NETDMP calls
NPXTHR	NPX throttled on transmit/ receive
NPXRCV	NPX unexpected receiver status
NPXCLR	NPX master circuit was cleared
NPXSEQ	NPX message out of sequence
NPXCON	NPX unknown circuit status
NPXRLS	NPX bad virtual circuit clearing
OUCREQ	Outgoing call request

OVERFL	NETBUF overflow entries
PWFAIL	Power fail checks
REMARK	Operator remark
RESET	Circuit resets
RING1	Tokens inserted into the ring
RING2	Ring dims out of receive blocks
RING3	Ring nodes not accepting transmits
RNGRCV	Spurious receive interrupt
RNGHRD	PNC hardware failure
RNGRES	Resource failure on communication queues
RNGTMT	Ring receive timeout
SHUTDN	Operator shutdowns
SMLC1	SMLC/MDLC status errors
SMLC2	SMLC/MDLC -- no STX preceding ETX
SMLC3	No system blocks for SMLC/MDLC protocol messages
SMLC4	SMLC/MDLC resets and CMDR data is printed
SMLC5	A CMDR has been sent
SMLC	Internal level 2 error
TIMDAT	Time and date entries
WARM	Warm starts

CONTROLLING THE SIZE OF THE NETWORK EVENT LOG FILE

If the network event log is allowed to grow indefinitely, it can consume large amounts of disk space. You can save space by doing the the following.

- Put a quota on the PRIMENET* directory. (The System Administrator's Guide tells you how to do this.)
- Print out, then delete old log files at regular intervals.

NETWORK EVENT MESSAGES

This section describes each network event that is displayed in the NETLST file, which is generated by both PRINT_NETLOG and LOGPRT -NET.

- BAD ENTRY: xxxxxx ... (OCT).

The program encountered an entry of unrecognized type or illegal length. An octal dump of the entry is provided for the number of words contained in the length field.

- CMDR SENT FOR LOGICAL LINE

An X.25 command reject was sent. The three-octet data field showing the cause is also displayed.

- CIRCUIT RESET - a ORIGINATED - controller: xx
  [CIRCUIT STATE: c (OCT) CAUSE: s DIAGNOSTIC: dddddd (OCT)]

A virtual circuit was reset. a indicates whether the origin of the reset was local or remote. xx indicates physical line or node. CIRCUIT STATE, CAUSE, and DIAGNOSTIC are only printed if a is REMOTE. s may be: DTE RESET, OUT OF ORDER, REMOTE PROCEDURE ERROR, NETWORK CONGESTION, or a word of octal data.

- COLD START [- PRIMOS REV rr ...]

A cold start of PRIMOS was performed. The PRIMOS revision number is indicated as well.

- *** END OF filename -- nnnnn ENTRIES, ppppp PROCESSED ***

This message is displayed when NETLOG reaches the end of the input file. nnnnn displays the decimal number of entries in the input file not including date and time and NETBUF overflow entries. ppppp displays the number of entries processed.

- ICS1.0 (X.25) - LINE NOT DEFINED
- ICS1.1 (X.25) - DECONFIGURE CODE WORD NOT QUEUED FOR LOGICAL LINE
- ICS1.2 (X.25) - LOGICAL CONNECTION DELETED ON LOGICAL LINE
- ICS1.3 (X.25) - LOGICAL CONNECTION NOT BROKEN, LOGICAL LINE
- ICS1.4 (X.25) - LCAD1_ NOT FOUND IN LCB FOR LOGICAL LINE
- ICS1.5 (X.25) - LOGICAL CONNECTION LOST FOR LOGICAL LINE

- ICS1.6 (X.25) - FLUSH TIMEOUT OCCURRED ON LOGICAL LINE
- ICS1.7 (X.25) - ILLEGAL FLUSH COMPLETE ON LOGICAL LINE
- ICS1.8 (X.25) - SYNCHRONOUS LINE NOT ASSIGNED, LOGICAL LINE
- ICS1.9 (X.25) - UNIDENTIFIABLE ERROR ON LOGICAL LINE

These messages are displayed if you are having problems with your Intelligent Communications Subsystem, Model I (ICS1). The X.25 notation is displayed only if you used the PRINT_NETLOG command.

- INCOMING CALL REQUEST

An incoming call request was received.

- INTERNAL LEVEL 2 ERROR FOR LOGICAL LINE

A fatal, internal error occurred to level 2. Two additional pieces of information are "ERROR CODE = xxx " and "LINE CONTROL BLOCK ADR = xxxx ".

- Level 3 network received a diagnostic packet

The following information is always displayed. One of the first two will be displayed depending on whether the second element in the network event buffer is equal to or greater than zero.

- Data Network Identification Code (DNIC) of the PDN is:
- This packet was sent by a DTE
- The controller number is: nnn
- The line number is: n

One of the following may also be displayed.

- No additional info, dcode is:
- Packet not allowed, dcode is:
- Packet on an unassigned Lchannel
- Packet too short, dcode is:
- Invalid GFI, dcode is:
- Timer expired, dcode is:
- Timer expired for CLEAR INDICATION

- Timer expired for RESET INDICATION
- Timer expired for RESTART INDICATION
- The diagnostic code is:

After one of these is displayed, the phrase "Diagnostic explanation" appears along with an octal string that represents an event in the network event buffer.

- LEVEL III PROTOCOL DOWN controller: xx

The Level III protocol for X.25 is down for this host. xx indicates physical line or node.

- LOCAL PROCEDURAL ERROR CAUSING CLEAR
  - controller: xx

A local procedural error caused the clearing of a circuit in this host. xx indicates physical line or node.

- NETBUF OVERFLOW -- nnnnn ENTRIES LOST

This indicates that nnnnn (decimal) event entries were lost due to overflow of the network event logging buffer (NETBUF).

- NETDMP CALLED AT: xxxxxx xxxxxx (OCT).
  [DATA: yyyyyy yyyyyy yyyyyy (OCT)]

A network software problem has occurred at this address. The routine NETDMP was called and asked to dump the three octal DATA words.

- NPX>R$CALL>R$CONN UNKNOWN CIRCUIT STATUS - NODE: xxxxxx (OCT).
  VIRTUAL CIRCUIT STATE (1): xxxxxx (OCT).
  VIRTUAL CIRCUIT STATE (2): xxxxxx (OCT).

PRIMENET has returned an unexpected status (error) code to NPX. This may be caused by the failure of a node in a controlled or uncontrolled way. A software failure was perhaps caused by a hardware failure.

- NPX>R$RLS ERROR IN VIRTUAL CIRCUIT CLEARING - NODE: xxxxxx (OCT).
  VIRTUAL CIRCUIT STATE (1): xxxxxx (OCT).
  VIRTUAL CIRCUIT STATE (2): xxxxxx (OCT).

There was a problem in clearing the virtual circuit (R$RLS). The returned VC status word 2 is not one of the existing XS$ status words.

- NPX>IRNRCV MASTER'S CIRCUIT WAS CLEARED - NODE: xxxxxx (OCT).
  VC STATE(1): xxxxxx (OCT). VC STATE(2): xxxxxx (OCT).

The connection between the master and the slave has been unexpectedly broken.

- NPX>IRNRCV MESSAGE OUT OF SEQUENCE IN BOUNCE DETECT.
  NODE: xxxxxx (OCT). MESSAGE SEQ#: xxxxxxx (OCT), NS: xxxxxxx (OCT).

NPX break detection and correction logic found a message out of sequence. NPX has failed, or data have been lost in the network.

- NPX>IRNRCV THROTTLED ON TRANSMIT OR RECEIVE -
  NODE: xxxxxx (OCT), MASTER/SLAVE FLAG: xxxxxx (OCT).

Network buffers are too full to send or receive an NPX message.

- NPX>IRNRCV UNKNOWN RECEIVE STATUS - NODE: xxxxxx (OCT).
  MASTER/SLAVE FLAG: xxxxxx (OCT). RECEIVE STATE: xxxxxx (OCT).

PRIMENET has returned an unanticipated status (error) code to NPX.

- OUTGOING CALL REQUEST

An outgoing call request was transmitted.

- PACKET OUT OF SEQUENCE - controller: xx CIRCUIT STATE: c
  SEQ # EXPECTED: d SEQ # FOUND: e

A packet was received with an unexpected sequence number.

- PNC HARDWARE FAILURE

The following messages appear when you encounter a PRIMENET Node Controller (PNC) hardware malfunction:

DMA FAILURE

NO SKIP ON INA

NO SKIP ON RECEIVE OTA

NO SKIP ON TRANSMIT OTA

There has been an apparent failure of the Prime Node Controller (PNC) hardware that controls the ring. The device has been shut down, and this node has removed itself from the ring. Hardware diagnostic tests should be run on the PNC.

- POWER FAIL CHECK

A power failure check has occurred.

- RESOURCE FAILURE RING QUEUE OVERFLOW

The following messages occur when you encounter an overflow on the ring.

DIM TO LEVEL II - RECEIVE PACKET LOST

LEVEL II to DIM - TRANSMIT PACKET LOST

LEVEL II TO DIM - TIMER PACKET LOST

- RING DIM OUT OF RECEIVE BLOCKS

The software controlling the Prime Node Controller (PNC) has been handling enough traffic to temporarily exhaust the available supply of buffers. If this event happens often, the system should be rebuilt with more buffers to handle this network's message load.

- RING NODE: node-number NOT ACCEPTING XMITS.

This event message means that one or more of the following occurred:

PACKET LOST, RING DOWN -- A transmit cannot get to the specified node.

PACKET WACKED, NODE HALTED OR CONGESTED — The specified node is in the ring but it is unable to receive from the rest of the ring.

NODE NOT IN RING -- The specified node has left the ring (the network was taken down).

If the reason is not one of those above, the error message will print out: TRANSMIT STATUS IS: xxxxxx (OCT). The octal content of the transmit status word from the PNC is xxxxxx.

- Ring node node-number Receive TIMEOUT - node down

This node has not received a packet from the specified node within the timeout period. The node is marked as DOWN in the STATUS USERS command.

- RING QUEUE OVERFLOW: DIM TO LEVEL II - RECEIVE PACKET LOST
  or
  LEVEL II TO DIM - TIMER PACKET LOST
  or
  LEVEL II TO DIM - TRANSMIT PACKET LOST

One of the queues used to move packets to and from the Prime Node Controller Device Interface Module (PNCDIM) has overflowed. Since the queues are designed to be large enough to handle peak traffic numbers, this message indicates a possible configuration problem. The packet being queued is returned to the free pool and ignored.

- SHUTDOWN BY OPERATOR

The operator issued a SHUTDN ALL command. This causes the network event log buffer to be automatically dumped.

- SMLC - NO STX PRECEEDING ETX. PHYSICAL LINE NUMBER = xxxxxx (OCT), DEVICE ADDRESS IS yyyyyy (OCT)

Packets sent on synchronous lines using BSC framing must begin with DLE/STX and end with DLE/ETX.

- SMLC RESET FOR LOGICAL LINE xxxxxx (OCT)

If the cause of this error was due to a command rejection, the CMDR data fields are displayed. Resets can be caused in six ways:

INVALID ADDRESS

COMMAND REJECT

INVALID NR

INVALID RESPONSE

INVALID NR ON REJECT

MAXIMUM NUMBER OF RETRIES EXCEEDED

- SMLC STATUS ERROR STATUS WORD IS xxxxxx (OCT)
  [PHYSICAL LINE # IS n]
  DEVICE ADDRESS IS yyyyyy (OCT), [NUMBER OF OCCURRENCES IS nnn]

An invalid status, xxxxxx, has been reported by the SMLC/MDLC. nnn is printed only on parity errors.

- SPURIOUS RECEIVE INTERRUPT ON PNC

A receive interrupt was issued by the Prime Node Controller (PNC) when no receive was pending. This indicates a hardware malfunction which disconnects the PNC from the ring. Hardware diagnostic tests should be run on the PNC.

- SYSTEM BLOCKS UNAVAILABLE FOR SMLC PROTOCOL MESSAGE
  MESSAGE IS xxxxxx (OCT), LOGICAL LINE NUMBER IS yyyyyy (OCT)

The level II synchronous protocol had no buffers in which to send the indicated type of protocol-generated message.

- "Text of operator remark"

Contents of the REMARK event, generated by use of the -REMARK option of PRINT_NETLOG.

- TOKEN INSERTED INTO THE RING NETWORK

The software controlling the PNC hardware issued a ring network control token.

- WARM START

A warm start of PRIMOS was performed.

PRINT_NETLOG ERROR MESSAGES

The following error messages may be displayed by PRINT_NETLOG:

- UNKNOWN CPU MODEL xx

A CPU model number was encountered with which PRINT_NETLOG is not familiar. PRINT_NETLOG generates a warning message and continues processing, treating the CPU model number as 0.

- BAD ENTRY ENCOUNTERED IN FILE 'logfile'

The event log file logfile contains an entry that is not defined by PRINT_NETLOG.

- DEFAULT INPUT FILE NAME NOT CONSTRUCTED

A file of the type NET_LOG.mm/dd/yy could not be found in PRIMENET* or the top level directories did not exist or the user had insufficient access.

- INPUT LOGGING FILE filename NOT FOUND.
  UFD <0>PRIMENET* CONTAINS LOGGING FILE FOR THE LATEST COLD START.
  ENTER INPUT FILE NAME (ENTER CR TO QUIT):

You did not specify an input event file and PRINT_NETLOG cannot find a file with a name of the format NET_LOG.mm/dd/yy. PRINT_NETLOG prompts you for the input filename.

OK TO DELETE FILE filename? ANSWER: 'Y' OR 'N'!

The output from PRINT_NETLOG may be directed to a file or to the terminal. If file output is desired, and PRINT_NETLOG finds that the output file already exists, this message is printed. The reply should be 'Y' to delete the file or 'N' to enter a new destination. If 'N' is entered, the message below is displayed.

NEW OUTPUT FILE NAME:

Enter a pathname. If you don't, PRINT_NETLOG will continue to query you with the preceding message sequence, asking if the existing output file is to be deleted or what will be the name of the new output file. Typing TTY sends output to the terminal.

- OUTPUT HAS BEEN PLACED IN FILE 'filename'

PRINT_NETLOG has completed the processing of the event entries and all other file manipulation that you requested. This message is generated if output was directed to a file.

22
Monitoring FTS

INTRODUCTION

This chapter explains how to monitor and maintain FTS. It assumes that FTS has been built, installed on the system, and then configured using the FTGEN utility. (Your System Administrator should refer to the Network Planning and Administration Guide for information on FTGEN.) This chapter describes the following operator responsibilities.

- Using the FTOP utility to start and stop FTS
- Starting the File Transfer Manager (YTSMAN) with FTOP
- Starting the file transfer servers with FTOP
- Managing user requests with the FTR utility
- Dealing with rush requests using FTGEN, FTR and FTOP
- Monitoring the FTSQ* directory
- Monitoring and archiving FTS system log files

THE FTOP COMMAND

The FTOP command is the FTS utility that allows you, as an operator, to start, stop, and monitor the operation of the file transfer servers. The FTOP command is available only if you are logged in as SYSTEM.

A server is a phantom process that handles file transfer requests for a single queue. Server processes must be started from the supervisor terminal. Each server can simultaneously handle as many as eight requests from a file request queue, of which there may be up to 9999.

The FTS manager phantom (YTSMAN) receives file transfer requests from remote sites, and passes them to appropriate local servers.

The format of the FTOP command is:

```
FTOP [option]
```

If option is omitted, FTOP displays a summary of the available options. Two options, -START_MNGR and -STOP_MNGR, apply to the FTS manager process, YTSMAN. All other FTOP options apply to file transfer server processes.

Summary of FTOP Options

The following list is a summary of FTOP options, which are fully described in the following section.

Option	Abbreviation	Purpose
-ABND_SRVR	-ASV	Abandon an FTS server process
-ABRT_SRVR_LINK	-ASVL	Abort an FTS server link
-HELP		Displays information on FTOP
-LIST_SRVR_STS	-LSVS	List server status
-START_MNGR	-STRMG	Start the FTS manager phantom process (YTSMAN)
-START_SRVR	-STRSV	Start an FTS server phantom process
-STOP_MNGR	-STPMG	Stop the FTS manager process
-STOP_SRVR	-STPSV	Stop an FTS server process

FTOP Options

All the options to the FTOP command are presented below in alphabetical order.

▶ -ABND_SRVR server-name

Abbreviation: -ASV

This option causes the specified file transfer server to immediately abort all current file transfers, placing the requests on hold, and to log out.

If the specified file server is not running, an error message results.

Note

The recommended way to stop a server is the -STOP_SRVR option. We do not recommend forced log out of a server.

▶ -ABRT_SRVR_LINK server-name link-number

Abbreviation: -ASVL

This option causes the specified file transfer server to abort the current file transfer on the specified link, placing the request on hold. The server continues running; it does not log out.

Each file transfer server may handle up to eight concurrent file transfers. To find the link number of an ongoing transfer, use the following command.

```
FTOP -LIST_SRVR_STS server-name
```

This command lists transfers, identifying each by its link number, in the range 1 to 8. If the file transfer server you specified is not running, or if the specified link is not active, an error message is displayed.

Note

You can accomplish the same effect by using the FTR -ABORT option. Use FTR -ABORT if you do not know the server name or link number, but you do know the request name or number.

▶ -HELP [subject]

This option displays information on the requested FTOP subject. To obtain a list of subjects on which help is available, type either

OK, FTOP -HELP SUBJECTS

or simply,

OK, FTOP -HELP

To obtain an FTOP command usage display, type either

OK, FTOP -HELP USAGE

or simply,

OK, FTOP

▶ -LIST_SRVR_STS [server-name]

Abbreviation: -LSVS

This option lists the status of the server you specified. It indicates the state of the actual server, that is, whether it is running or not, and lists the state of each of the eight possible file transfers that the server may be running. Each transfer is identified by a link number, in the range of 1 to 8. If no server name is specified, the status of all the configured servers is listed.

For active server links, the pathname of the local file will be displayed, stripped of any passwords.

If local requests do a send operation, the pathname is the source file pathname, and if the request is a get or fetch operation, the pathname is the destination file pathname.

For requests initiated remotely, the display will be either the pathname of the local file being fetched/sent or the device type (LP) to which the remote file is being sent.

In addition, for both local and remote requests, the -LIST_SRVR_STS command displays the following information.

- Whether the file is being sent (S) or (fetched) (F).
- The remote site. This information is not always available to the local server. In that case, 'REMOTE' is displayed.
- The start time of the file transfer, expressed in 'hh:mm'.

Below is an example of the new display for local and remote requests:

```
Link. Request.   Start time. Status.          Remote site.

1   2       S    17:02    Active.            BIRCH
    Local file   - <TSTDSK>TEST>SEND_FILE1
2   3       S    17:04    Active.            ELM
    Local file   - <TSTDSK>TEST>SEND_FILE2
3   4       F    17:05    Active.            LINDEN
    Local file   - <TSTDSK>TEST>FETCH_FILE1
4   5       S    17:01    Active.            FIR
    Local file   - <TSTDSK>TEST>SEND_FILE3
5   REMOTE  S    16:55    Active.            ASH
    Local device - LP
6   REMOTE  F    17:02    Active.            OAK
    Local file   - <TSTDSK>TEST>REMOTE_FETCH1
7   REMOTE  S    16:59    Active.            PINE
    Local file   - <TSTDSK>TEST>REMOTE_SEND1
8   REMOTE  S    17:02    Active.            WILLOW
    Local file   - <TSTDSK>TEST>REMOTE_SEND2
```

▶ -START_MNGR [manager-name]

Abbreviation: -STRMG

This option starts up the FTS manager phantom. The default manager name is YTSMAN.

Note

The command FTOP -START_MNGR should be invoked only from the supervisor terminal. Attempting to use this command from any other terminal results in an error message.

You can add the command to start up the manager to the PRIMOS cold start PRIMOS.COMI file. See the System Administrator's Guide for more information.

▶ -START_SRVR server-name

Abbreviation: -STRSV

This option starts up the specified file transfer server as a phantom. If that server is already running, an error message is displayed.

The command FTOP -START_SRVR should be given only from the supervisor terminal. Doing so insures that the server phantom will be created with a username of the server and that the process priority and timeslice will be automatically set in accordance with the configuration of the server.

Invoking this option from a terminal other than the supervisor terminal would result in the specified server not running under its proper username and the server's configured priority and timeslice not being set. The system defaults for these values would be assigned instead.

It is particularly important in an ACL environment that the server phantom run with the configured server name, since users will have to grant that particular server name appropriate access rights to their directories to and from which files are to be transferred.

The invocation of this option from a terminal other than the supervisor terminal would result in the specified server not running under its proper user name and the server's configured priority and timeslice not being set. The system defaults for these values would be assigned instead.

You can add the commands to start up the required file transfer servers to the PRIMOS cold start PRIMOS.COMI file. See the System Administrator's Guide for more information.

▶ -STOP_MNGR

Abbreviation: -STPMG

This command causes the FTS manager to complete its current work and log out. If the manager is not running, an error message is printed.

Stopping the FTS manager prevents remote requests from being received and serviced. If one or more transfer server phantoms are running, then local requests are processed as usual. However, remotely-initiated transfers of files to and from the local site will not succeed, since the FTS manager phantom will not be present to receive these incoming calls from PRIMENET and pass them on to the appropriate file transfer server. Such requests will be retried by the remote file transfer server at 30-minute intervals.

▶ -STOP_SRVR server-name

Abbreviation: -STPSV

This option causes the specified file transfer server to complete the file transfers that are currently in progress, and then log out. If the server is not running, an error message is displayed.

This is the recommended way to close down an FTS server.

Even if a server is not running, users can still submit requests, which are queued.

STARTING, STOPPING, AND MONITORING YTSMAN

YTSMAN is the FTS default name for the FTS file manager phantom process. You can provide another name with the START_MNGR command, described below. You can monitor the FTS manager phantom process with the STATUS USERS command.

YTSMAN receives file transfer requests from remote nodes and passes them to the appropriate local file transfer servers. When running, it maintains a command output file with the following pathname.

```
FTSQ*>YTSMAN.COMO
```

Both the FTS manager and the FTS file server must be started from the supervisor terminal. You can incorporate the following command into the PRIMOS.COMI file so that the file transfer manager starts automatically at each cold start.

```
FTOP -START_MNGR [manager-name]
```

The FTS manager phantom maintains a command output file when it is running. The command output file, FTSQ*>YTSMAN.COMO, is generated by the manager.

To stop the manager, type the following command at the supervisor terminal.

```
FTOP -STOP_MNGR
```

See the FTOP section earlier in the chapter for more information on these FTOP manager commands.

STARTING, STOPPING, AND MONITORING FILE TRANSFER SERVERS

File transfer servers are the phantom processes that handle file transfer requests. The System Administrator configures file transfer servers and assigns server names.

File transfer servers are phantom processes. FTS server phantoms maintain log files in the FTSQ* directory while they are running. The names of these log files are configured using FTGEN, so list the appropriate server configuration using FTGEN's LIST_SERVER command to discover the log filename. Operators can monitor a command output file called FTSQ*>COMO.FTS>server-name, which is maintained by active FTS server phantoms.

The operator uses the FTOP command to start, stop, and monitor the operation of the file transfer servers. FTOP is available only to the operator logged in as the user-id SYSTEM. FTOP command options are fully described earlier in this chapter.

To start up an FTS server process, enter the following command.

```
FTOP -START_SRVR server_name
```

The above command is normally included in the PRIMOS.COMI file on your system.

To obtain a list of all configured servers, including the status and user number of each server, enter the following command.

```
FTOP -LIST_SRVR_STS
```

A few minutes before shutting down the system, the FTS servers should be told to shut down as soon as they complete any transfers in progress. To do this, enter the following command for each running FTS server. (You must be logged in as SYSTEM.)

```
FTOP -STOP_SRVR server_name
```

Stopping the servers in this manner ensures that all transfers currently in progress will be successfully completed before the log out.

As each server shuts down, it sends a message to the supervisor terminal.

Sometimes, it may be necessary to immediately shut down the FTS servers, even if they are currently transferring files. This is known as "abandoning" the FTS servers. When an FTS server is abandoned, it places any file transfers it is currently processing on hold in the queue, so that they can be started up again later. It then logs itself out. To abandon an FTS server, enter the following command while you are logged in as SYSTEM.

```
FTOP -ABND_SRVR server_name
```

Note

When the local FTS server and YTSMAN are not running, local users may still queue requests with FTR.

MANAGING AND MONITORING USER REQUESTS

While it is the responsibility of users to track their own requests, you should check the general status of requests with the FTR -STATUS_ALL option. You should watch for requests that have been repeated many times or that have been put on HOLD for a long period of time with either the above command or with the FTR -DISPLAY command option.

Typical causes for problems with file transfer requests are:

- Network congestion.
- The remote site is down.
- The remote server or manager has not been started.

One example of an FTR management option is to abort a request that is already in progress. To do this, enter the following FTR management command.

```
FTR -ABORT request-name
```

The specified request will be put on hold. Later, when you are ready to release the request and allow the transfer to take place, enter this command.

```
FTR -RELEASE request-name
```

Other FTR management options that you can use to control user requests are described in Chapter 6.

RUSH REQUESTS

If you want to rush the transfer of a particular request, you need to use a combination of the FTGEN, FTOP, and FTR commands. See the Network Planning and Administration Guide for more information on the FTGEN command. New requests queued during this procedure are not held. If this causes all free server links to be used up, use the following procedure.

1. Block the file transfer queue on which the rush request(s) are queued. Use the FTGEN BLOCK_QUEUE command to prevent other requests from being added to the queue.
2. Use the FTR -HOLD command to suspend all waiting requests.
3. Use the FTOP -LIST_SRVR_STS command to discover if there are sufficient free server links available, out of the maximum of eight, to process the request(s) you want to rush.

4. If necessary, use the FTOP -ABRT_SRVR_LINK command to put requests on hold. This frees the server links of currently transferring files.

5. Use the FTR -RELEASE command to release the rush request(s).

6. When all rush requests have been completed, use the FTR -RELEASE command to release all requests you had put on hold.

7. Finally, use the FTGEN UNBLOCK_QUEUE command to allow new submissions of file transfer requests to previously blocked queues.

Note

Normally, you might need only the -HOLD and -RELEASE commands. The steps required depend on the activity of the FTS server when you want to rush one or more requests.

MONITORING THE FTSQ* DIRECTORY

The FTSQ* directory holds copies of user files that are to be transferred as well as FTS log files. You should make sure there is adequate disk space available to accommodate these copies. Use the LIST_QUOTA and AVAIL commands, described in the System Operator's Guide, Volume II, for information on monitoring disk space utilization.

MONITORING AND ARCHIVING FTS LOG FILES

FTS log files are maintained in the FTSQ* directory. The file names are specified by the System Administrator as part of the FTS configuration. Server log files record all events for incoming and outgoing file transfers, and can be useful in providing a record of FTS usage when you are tracking the progress of a particular request. You should examine the server log daily to check the status of FTS. This can be done simply by using an editor like ED to locate the current date in the file, and then locating RESULT. The operator can use ED's X command to repeat the LOCATE RESULT command line.

For example:

```
OK, ED FTSQ*>FTP.LOG
EDIT
L DECEMBER 1
00.00.18: [1.1] Request MEMO (53) started Thursday December 1, 1983
L RESULT
00.00.19: [1.1] RESULT: Transfer Aborted : Out of order.
X
9.31.59: [2.1] RESULT: Transfer Rejected: File not available.
X
9.32.50: [2.1] RESULT: Transfer Rejected: Problem with remote file.
X
9.38.08: [4.1] RESULT: Transfer Terminated: Satisfactory and Complete.
```

Log files are not limited in size, and should thus be regularly archived so that the FTSQ* directory does not become full.

STOPPING FTS

Before a STOP_NET command is given, you must stop FTS by using the FTOP -STOP_SRVR and the FTOP -STOP_MNGR commands to shut down the file servers and the file manager (YTSMAN). Once the servers and the manager have stopped, then the STOP_NET command can be used.

Once the network has been started again with the START_NET command, you can restart the servers and the manager with the FTOP START_SRVR and FTOP START_MNGR commands.

23
Monitoring RINGNET

RING DIAGNOSTIC PROGRAMS

Two programs let you see what is happening in RINGNET. The first program, MONITOR_RING, gathers and displays error statistics throughput. The second program, FIND_RING_BREAK, shows you the location of a break in the ring.

The first part of this chapter describes how RINGNET operates. The second part of the chapter explains the MONITOR_RING program, how to use it, and how to interpret its statistics. The remainder of the chapter similarly describes the FIND_RING_BREAK program.

RINGNET OVERVIEW

This section describes the following subjects.

- RINGNET hardware
- RINGNET terminology
- How RINGNET works

You should also read Chapter 10, which describes PRIMENET architecture, before you use the ring diagnostic programs.

RINGNET Hardware

RINGNET is Prime's Local Area Network (LAN), which consists of nodes connected through a high-speed, one-way, serial-synchronous twin-axial cable.

Each node in a ring network is equipped with

- A PRIMENET Node Controller (PNC) board, which controls the ring protocol and the flow of data between ring nodes.
- A junction box, which connects the twin-axial cable to the computer. The junction box has a passive relay that switches to "pass-through" if it detects a power failure on the PNC. This preserves a ring's integrity.

Each PNC acts as an active data repeater for packets between other ring nodes. In other words, a PNC transceives all packets passing through it.

RINGNET Terminology

The following glossary presents some RINGNET software and hardware terminology that is used in this chapter.

ACK byte	This byte is in the trailer portion of the data packet and is set by the PNC of the receiving node to indicate that the message was successfully received. It can also be set to indicate invalid data by any active PNC (refer to the definition of parity check below).
active data repeater	A device that amplifies a signal; the PNC is an active data repeater because it transceives packets automatically.
bit	An acronym for binary digit: There are eight bits to a byte.
byte	Eight bits of data.
data field	The portion of the data package that contains the actual data in protocol format.
CRC	Circular Redundancy Check. A check performed by a PNC on every data packet it transceives. If a packet fails this test, the PNC sets the ACK byte accordingly.

event	A significant system or network occurrence such as a cold start, machine check, disk error, and network link problems.
header	The portion of a data packet that contains information such as the ids of the target and sending node or packet type.
LAN	Local Area Network. A method of linking a group of independent computer systems.
leading frame	A bit pattern that tells the PNC that a data packet follows.
Level 1	PRIMENET's hardware interface. This level (or layer) acts as an intermediary between the physical transmission medium (twin-axial cable or transmission line) and Level 2.
Level 2	PRIMENET's link protocol level. It describes a protocol for linked systems to adhere to when transferring data between themselves.
Level 3	PRIMENET's packet interface. It creates and controls connections across the network, handles error recovery, and controls the flow of information.
NETLOG	A log file that records network events. The PRINT_NETLOG command produces a copy of this file.
node	An independent computer system that is part of a network.
packet, broadcast	A 10-byte packet periodically sent by a node that contains the node's ring id; this packet is received by all active PNCs.
packet, data	A package that contains packet protocol and data. It consists of a leading frame, header, a data field, and a trailing frame. The size of the data packet can be set by the System Configurator through CONFIG_NET.
parity bit	A bit whose value indicates whether an ACK is good or corrupt. On the ring, this bit is contained in the ACK byte and is set by the PNC.

parity check	A check performed by the PNC to determine if the ACK byte for a data packet is good or corrupt. The PNC sets the parity bit in the ACK byte accordingly.
PNC	The PRIMENET Node Controller. PRIMENET hardware that controls ring protocol and the flow of data between nodes on a ring.
PNCDIM	The PNC Interface (software) Module.
RINGNET	Prime's Local Area Network.
ring node id	A number from 1 to 247 that identifies, and is unique to, a particular node.
trailer	The portion of the ring packet that contains information such as the ACK byte and the trailing frame.
trailing frame	A special bit pattern that tells the PNC that the end of the data packet has been reached.
transceive	To receive and transmit data simultaneously. The PNC is a transceiver that uses a 4-bit time delay between reception and transmission of data.

The following terms are used to described conditions on a ring.

inserted token	If a token is lost, an active node inserts a token into the ring to allow communication between nodes to continue uninterrupted.
lost token	Infrequently, a token is lost because of interference or noise on the ring, usually from a node entering or leaving the ring.
multiply accepted	A node has transmitted a packet and has had more than one node accept it. Because node ids are unique, this should not occur.
NAK	Negative Acknowledgement. Indicates that a data packet failed CRC, or an ACK byte did not pass parity checking. When the node that sent the packet receives the NAK, it retransmits the data.
node is down	When node A has never heard from or cannot transmit to node B, node B is considered down.

node is up	When node A can receive from node B, node B is up.
NON ACKNOWLEDGED	A packet sent by the transmitting node returns with the ACK byte unchanged. This indicates that the target node is not physically in the ring. No attempt is made to retransmit.
NOT RETURNED	The transmitting node sees a token before it sees the packet it sent; this is an indication of corrupt data. The node retransmits the data when this occurs.
time out	The transmitting node saw neither a token nor the packet within a certain time period. The ring may be broken or going through token recovery. The node waits for a token and retransmits the packet. If a second time out occurs for the same packet, no further attempt to retransmit is made.
token recovery	The mechanism PRIMENET uses to replace a lost token.
WACK	Wait Acknowledge. The receiving node "acknowledges" the packet, but does not have a buffer free to receive it. The transmitting node retransmits the packet.
WACK linked list	A linked list of WACKed packets in node number sequence (in time sequence within a node number). The PNCDIM maintains this list and attempts to deliver these packets a set number of times before marking the receiving node as down and aborting the packets.

How RINGNET Works

RINGNET uses a token ring protocol. A special bit pattern called a token circulates counterclockwise continuously around the ring. A node cannot transmit data until it detects the token. When a node is prepared to transmit data and detects the token, the PNC

- Strips the token from the ring
- Issues a packet containing a 4-byte header and from 4 to 2044 bytes of data
- Immediately returns the token to the ring behind the packet

The packet circulates around the ring, at a data rate of eight megabits per second, to the destination node. As it passes through the PNCs of other nodes, each PNC performs a CRC and parity check to verify the integrity of the data. The PNC sets the ACK byte accordingly if there has been an error.

The packet arrives at the destination PNC, which does the following:

- It checks the header to find out if it is the destination node.
- It performs a CRC and parity check.
- It sets the ACK byte to indicate whether the data packet is good or corrupt. (If the data packet is corrupt, the ACK byte is set accordingly, and the packet is not accepted; it is returned to the transmitting node.)
- It accepts the packet (if the data packet is good) into a receive buffer while it is transmitting it in the ring.
- It notifies upper level software that a packet has been received.

The packet travels around the rest of the ring back to the source PNC, which

- Removes the packet from the ring
- Passes on the token
- Checks the acknowledgement flag
- Interrupts the source operating system to signal transmission completion

THE MONITOR_RING PROGRAM

The MONITOR_RING program gathers and displays statistics about communication on the ring. It does not monitor the signals on the ring itself; it monitors ring traffic on the node from which you run it. You examine this information through the use of three screens: the Basic Display, Error Display, and Trace Display. The Basic Display monitors the normal functioning of the node; it monitors packets, bytes, and broadcast packets received and transmitted. It offers an updated display at configurable intervals. The Error and Trace Displays are detailed snapshots; they elaborate on the data in the Basic Display. The Basic Display monitors the normal functioning of the node.

By using the monitor, you can determine if the ring is working properly, if a situation exists that requires further analysis, or if the ring is malfunctioning. The monitor also gathers statistics on transient errors.

This section explains how to invoke MONITOR_RING and how to select the different displays. The section INTERPRETING RING STATISTICS describes the displays in detail and offers advice and guidance on interpreting the information.

Invoking the MONITOR_RING Program

The options with MONITOR_RING are listed and described briefly below.

MONITOR_RING [options]

-HELP -H	This option displays the syntax and a summary of options.
-TTP type	Use this option to specify the terminal type. Choices are NO (no terminal output), FOX, OWL, TTY (any scroll mode terminal), PT45, PST100. The default is PST100.
-FREQ f	Choose the number of seconds between updates of the Basic Display counters. The number must be an integer. The default is two seconds.
-TIMES n -TI	If you wish to update the Basic Display counters only a specific number of times, use this option. Otherwise, MONITOR_RING obtains data samples every f seconds for as long as you run the program.
-RESET_DAY -RD	This option resets all counts at midnight. The default is no reset. All counts are carried forward.
-RESET_HOUR -RH	This option resets peak and cumulative counts on the hour. The default is no reset. All counts are carried forward throughout the duration of the program.
-REPORT m filename -RPT	Use this option to write the dynamic data to file "filename" each minutes. The default is that no file is written. You must specify a value for m. An example of this file appears later in this chapter.

-INPUT filename -I	This option instructs the program to take input from file filename instead of using dynamic data. The default is the program taking the data dynamically.
-TRACE -TR	This option includes inter-node data in the REPORT file.

Selecting MONITOR_RING Displays

Once in MONITOR_RING, you can move easily among the three displays by using the seven single-key commands listed below.

Command	Result
B	Displays the dynamic Basic Display
E	Displays the static Error Display
F	Freezes the display
H	Displays an explanation of the single key commands
Q	Exits from the program and returns you to PRIMOS
RETURN	Advances to the next Trace Display
T	Displays the first Trace Display

If the F key is pressed, the display is frozen indefinitely. You can re-invoke the B command to refresh the display.

INTERPRETING RING STATISTICS

MONITOR_RING offers you three different ways to examine RINGNET information. The Basic Display is shown in Figure 23-1.

The remaining two displays, the Error Display and the Trace Display, provide greater detail about a particular event or aspect of ring activity. This section describes each of the displays, the type of data shown, and how to interpret the data.

```
MONITOR_RING Version 1.0
Node: (nnn)  MMM DD YYYY HH:MM:SS        Freq (f)          Start HH:MM:SS
                          Period         Peak Period       Cumulative        Total
Packets Transmitted       xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Packets Received          xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx

Bytes Transmitted         xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Bytes Received            xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx

Inserted Tokens           xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Receive Errors            xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Transmit Errors           xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Multiply Accepted         xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Multiple Tokens           xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx

Wacked Transmits          xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Aborted Transmits         xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx

Non Returned Packets      xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Non Acknowledged          xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx

Broadcast Transmitted     xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
Broadcast Received        xxxxxxxxxx     xxxxxxxxxx HH:MM:SS  xxxxxxxxxx     xxxxxxxxxx
```

nnn	The node name of the local node
MMM DD YYYY	Month, day, and year of the data displayed
HH:MM:SS	The time of day in hours, minutes, and seconds
f	The frequency of update in seconds
xxxxxxxxxx	An n-digit counter

Basic Screen
Figure 23-1

Basic Display

The Basic Display shows statistics for every data and control packet sent or received, shows the amount of activity on the node, and indicates the status of the ring and node. The first line of the display identifies the program and the version. The second line displays the following information.

Heading	Meaning
Node: (nnn)	The node on which the program is (or was) running.
MMM DD YYYY HH:MM:SS	The date and time of the most recent sample. If the data are being read from a file by use of the -REPORT option, the display shows the time at which the data were collected.
Freq f	The frequency at which the data are collected (set by the -FREQ option). If the data are being read from a file, this tells what f was set at.
Start HH:MM:SS	The time RING_MONITOR was started.

The third line describes the contents of the four vertical columns.

Heading	Meaning
Period	The count for the last period of length f. If dynamic data are being collected, then this column will be refreshed every f seconds.
Peak Period	The maximum count that has occurred within any period since monitoring began. The period is defined by the value of Freq f. For example, if Freq f is 2, then the value shown for each field represents the highest count that occurred during a two second period while the program was running. The time stamp to the right of the count indicates when this peak count occurred.
Cumulative	The total of the counts since the start of the program. If the option -RESET_HOUR is functioning, then this column shows the totals accumulated since the beginning of the hour.
Total	The grand totals since the network was started since the last START_NET command. If the option -RESET_DAY is specified, then the grand totals from midnight are shown.

The leftmost column of the Basic Display tells what fields are being sampled. Table 23-1 summarizes the function of each of these fields, and detailed descriptions follow.

Table 23-1
Fields in the Basic Display

Packets Transmitted Packets Received	A count of all successful transmitted and received packets.
Bytes Transmitted Bytes Received	A count in data bytes.
Inserted Tokens Receive Errors Transmit Errors Multiply Accepted Multiple Tokens Wacked Transmits Aborted Transmits	These fields show a count of various errors on the ring. Normally, the count is low or zero. Non-zero values do not always indicate a problem.
Non Returned Packets	This field displays a count of packets not returned and assumed lost.
Non Acknowledged	This field displays a count of packets that were not received at the destination.
Broadcast Transmitted Broadcast Received	These fields show a count of broadcast control packets. Each node periodically sends out status information to every other node in a broadcast packet.

Packets Transmitted: This field shows the number of packets successfully transmitted by the local node. This count does not include any transmitted packets that are being counted as errors, for example, Transmit Errors. This figure does, however, include transmitted broadcast packets, since these are successful transmits.

All the traffic transmitted from the local node, regardless of errors, would be a total of the following counts.

Packets Transmitted
Transmit Errors
Multiply Accepted
Multiple Tokens
WACKed Transmits
Non Returned Packets
Non Acknowledged

Packets Received: This field shows the number of packets successfully received by the local node, including the count of received broadcast packets (since they were successfully received). As with Packets Transmitted, the count does not include Receive Errors.

The count of Packets Received (and Broadcasts Received, see below) will include any broadcast message transmitted by the node itself. The PNC will see that the packet is a broadcast message and will receive it. The Transmitted Broadcast packets are therefore counted as both transmitted and received.

Bytes Transmitted: This field displays the count of the bytes contained in the Packets Transmitted.

Bytes Received: This field shows the count of the bytes contained in the Packets Received.

Inserted Tokens: This field shows the number of tokens inserted into the ring by the node. If the local node has a transmit pending and does not see a token within a reasonable amount of time, it will assume that the token is lost and insert a new token into the ring.

Extra tokens will be inserted when traffic cannot circle the ring. This may indicate that there is a break on the ring. Run FIND_RING_BREAK to determine if and where the ring is broken.

Occasionally, an extra token is inserted into a properly functioning ring. For example, the operation of the ring is disrupted for about 10 milliseconds when a node enters or leaves it. This amount of time is long enough to cause the token to be stripped off of the ring. The RINGNET hardware and software recognize that the token has been removed and act immediately to maintain uninterrupted operation. One of the nodes reinserts the token. In this case, an occasional inserted token is acceptable and indicates that the ring is functioning normally.

A high count may indicate that there is a problem with the PNC. The PNC may not recognize a token, or the PNC of another node may be malfunctioning and inappropriately removing the token from the ring. Run FIND_RING_BREAK and check to see if other nodes are having the same problem to determine if there is a problem with a PNC.

Receive Errors: This field shows the number of packets received in error by the node. Normally, the count is zero. Use the Receive Errors status codes shown in the Error Display to determine the cause.

Transmit Errors: This field indicates corrupt data. This can be caused by a CRC validation failure. Also, a packet that did not return before the next token was seen can cause this error. These are explained in more detail in the Error Display.

Receive Errors and Transmit Errors: These fields show errors that are caused by the degradation of the data as they are sent along the ring. Because the ring is an extremely reliable medium, these errors are rare. The four factors that can cause this type of error are

1. A cable that is longer than 750 feet between active nodes
2. A loose connection on the ring
3. Noise generated by a node entering or leaving the ring corrupting the data portion of the packet and/or destroying the token
4. Faulty PNC hardware

Normally, a single count per incident indicates that a node is entering or leaving the ring. A multiple count per incident usually points to a hardware problem. These counts usually are very low in proportion to the traffic on the ring.

Multiply Accepted: This field indicates the number of times the node transmitted a packet and had more than one node accept it. Because node ids are unique, this should not occur. A node will automatically disconnect from the ring if at start-up it finds that another node has the same ring id.

Note

Broadcast packets are not part of the count. They are expected to be accepted by all nodes.

Multiple Tokens: This field indicates that there have been multiple tokens on the ring. There is supposed to be only one. If the count is not zero, then the following has happened.

A node that is preparing to transmit has seen the token and taken it off the ring. Before the node has completed transmitting, another token is received by the node. This node's PNC strips this extra token off the ring. Thus, the PNC removes the extra token before it can interfere with the node's transmit operation.

WACKed Transmits: This field shows the number of times that the receiving node was unable to accept a packet sent by the local node because its receive buffer was already full. The PNC retransmits the packet up to 20 times. If the packet still is not accepted, it is placed on the WACK list. The PNC then tries to deliver the packet up to five more times. Each attempt is one second apart and consists of up to 20 retransmissions. If this fails, the packet will be aborted and the receiving node is marked as down.

Usually, this is a transient failure. It may be, however, that the target node has halted without master clearing the PNC. Master clearing the PNC restores proper ring operation.

Aborted Transmits: This field shows the number of aborted packets. An aborted packet is one that could not be delivered successfully. Usually, the packet is aborted because the number of retries in the WACK procedure has been exhausted. However, it is possible that a packet will be negative acknowledged (NAKed) or multiply accepted 20 times and then aborted.

Non Returned Packets: This field shows the number of times the node sent out a packet and saw neither it nor the token return within a timeout period. The timeout period is long enough so that on an unbroken ring this behavior should not occur.

If the node sees neither the packet nor a token within the timeout period twice in a row, the packet is aborted and the node is marked down. If this happens to three packets in a row, the supervisor terminal displays the message:

****** RING MAY BE DOWN ******

The message is followed by a date and time stamp. Usually, this error occurs if there is a break in the ring. Use FIND_RING_BREAK to determine if there is a break and what its location is.

Non Acknowledged: This field displays the number of times the node has transmitted a packet and had it returned without any indication that the target node is on the ring. This is the result of configured nodes not being physically in the ring.

Broadcast Transmitted: This field counts how many Broadcast Packets were sent by the node. A Broadcast Packet has a special node-id and is accepted by all nodes ready to receive. A Broadcast Packet is used to send the node status message from one node to all other active nodes on the ring. This message identifies the node and indicates that it is on the ring and able to communicate.

Broadcast Received: This field shows the number of broadcast packets received, including any sent by the node being monitored.

Note

The Packets Transmitted and Packets Received fields include Broadcast Transmitted and Received packets.

Error Screen

The Error Display (Figure 23-2) elaborates on error-related data that are shown in the Receive and Transmit fields of the Basic Display. It indicates possible problems along the ring. The Error Display heading includes an identifier, "Error Display", and a reminder of some of the single-key commands: H, for the Help menu; B, to invoke the Basic Display; and T, to invoke the Trace Display.

```
RING IS {UP, DOWN, DISABLED, NOT CONFIGURED}     Error Display
Receive Errors                  x                ((commands: H for help, B, T))
Receive Error Status         oooooo  [description of errors]
Receive Error Node       (node id)  (node name)
Receive Bad Protocol            x

Transmit Errors                 x
Transmit Error Status        oooooo  [description of errors]
Spurious Transmit Interrupt     x
Transmit WACK Node       (node id)  (node name)
Duplicate Node           (node id)  (node name)

No Receive Buffer               x                Receive Queue Count       x
Receive Queue Full              x                Transmit Queue Count      x
WACKed Packets                  x                WACK List Entry Count     x
Aborted Packets                 x                WACK List Max Entry       x

Fatal Errors:  PNC INA Failed                    x
               PNC OTA Received Failed           x
               PNC OTA Transmit Failed           x
               PNC DMA Failed                    x
               Spurious Receive Interrupt        x

  oooooo  The octal value of the status word
       x  The appropriate count
```

Error Screen
Figure 23-2

Status Line

The Error Display status line defines the status of the ring as seen from the local node.

RING IS {UP, DOWN, DISABLED, NOT CONFIGURED}

UP	The ring is up if the local node is successfully communicating on the ring.
DOWN	The ring is down if the local node is in cold start state and has not yet been brought up.
DISABLED	This status can mean one of two things. First, the START_NET command is unable to start the network. This failure is caused by a configuration or hardware error. Secondly, the RINGNET is considered disabled if, during normal operation, a fatal error occurred. The Fatal Error field in the Error Display indicates which situation has occurred. A record of fatal network errors appears in the network event file, which is described in Chapter 21.
NOT CONFIGURED	The ring is not configured when the ring or node number is not included in the configuration file.

Error Display Fields

The fields in the Error Display are explained in detail below.

Receive Errors: This is the same counter that was shown in the Basic Display.

Receive Error Status: The octal value of the status word is displayed. Bits are numbered 1-16 from the left, and have the following significance when set to 1.

Bit	Description
12 to 16	Not used.
11	DMA: End of Range before End of Buffer. This is a DMA transfer error; the CPU memory buffer became full while there was still data coming in. One possible cause is a bad PNC board. Alternatively, the buffer size of a system may have been changed through the configurator. If the node with the larger buffer transmits to the node with the smaller buffer, this error occurs.
10	NONE: Receive Busy; not displayed.
9	BUF_PAR: PNC Receive Buffer Parity Error.
8, 7	ACK_PAR: Indicate parity errors in a received ACK Byte.
6 to 5	Not used.
4	CRC: Bad CRC. This bit can be set by any node in the ring while the packet is being transceived.
3	WACK: Wait Acknowledge. An earlier node recognized this packet as addressed to it, and verified the CRC as good. However, it was not able to accept the packet due to the lack of a receive buffer available in the PNC.
2	MULT_ACK: Multiple Acknowledge. An earlier node recognized this packet as addressed to it; however, bit 1 of the ACK Byte had already been set by another node.
1	PREV_ACK: Previous Acknowledge. An earlier node recognized this packet as addressed to it.

Note

If the bits 3, 2 or 1 are set in a packet that is not a broadcast packet, there may be a configuration error in the ring. Possibly, two or more nodes have the same node-id.

Receive Error Node: This field identifies which node transmitted the packet that was received in error. The node-id and node name are shown.

Receive Bad Protocol: This field displays the number of packets received that have unrecognizable protocol bytes. Two protocol bytes are allowed. The first is for the broadcast node-id packet, which has a protocol field of 0. The second is X.25 level III, which has a protocol field of :1000.

Any packet with a different protocol field is discarded.

Transmit Errors: This is the same counter that was shown on the Basic Display.

Transmit Error Status: This field shows the octal value of the status word, and a brief description code.

Bit	Description
13 to 16	Not used.
12	ACKB_or_CRC: This code indicates that a packet was returned with an ACK byte check or a bad CRC indication.
11	PKT_DNR: Packet did not return. A token was received after completion of the transmit and before the return of the packet header. These packets are counted in the Transmit Error field.

Note

The Non Returned Packet field shows a different error, where neither a packet nor token is received, and the transmit time out expires. A value greater than 0 in the Multiple Token Field is an indication that a second token has arrived before the completion of a transmit.

10	Not Printed: This code indicates Transmit Busy condition (not an error).
9	BUF_PAR: Transmit Buffer Parity Error.
8,7	ACK_PAR: These codes indicate parity errors in a returned ACK Byte.
6	Not Displayed.
5	Unused.

4 CRC: Bad CRC. This bit can be set by any node in the ring. The CRC validation is done automatically during transceive.

3 WACK: Wait Acknowledge. The target node was not able to accept the packet because of receive congestion; however, the data were valid (no CRC error).

2 MULT_ACK: Multiple Acknowledge. More than one node saw this packet as addressed to them. Unless this packet is a broadcast packet, a configuration error exists.

1 ACK: This is not an error code. It indicates that this packet was accepted by the target node. This code is used in conjunction with the Multiple ACK or WACK fields to detect duplicate nodes.

Note

If bit 2 is set, or if bit 1 and bit 3 are set, then the packet was accepted by more than one node. This is only acceptable for a broadcast packet. If the packet is a broadcast packet, it is not marked as an error. However, the transmit status bits will be shown, and the Multiple ACK, or Previous WACK, comments may show without any errors being indicated.

Transmit Wack Node: This field identifies the node to which the most recent aborted packet was addressed. The node-id and node name are shown.

Often, the identified node may have halted without doing a master clear and is therefore WACKing all packets addressed to it. Once a master clear is performed, that node can successfully transmit and receive packets.

Spurious Transmit Interrupt: This field shows the number of times that the local node has received a transmit interrupt from the PNC without a previous transmit packet command. However, this can occur when the node tells the PNC to issue a token, and is not indicative of any problem on the ring.

No Receive Buffer: This counts the number of times that the RINGNET software attempts to allocate a buffer for receive, but one is not available.

Receive Queue Count: The receive queue is the data queue from the PNCDIM (lowest level software) to the Level 2 protocol module, RNGRCV. This holds buffers of data that have been received by the PNC (or transmit buffers that have been aborted by the PNCDIM). If the cumulative count in this counter indicates that most of the receive buffers are sitting in this queue, it implies that the higher level module is not able to process the packets as quickly as they are being received.

Transmit Queue Count: The transmit queue is the queue from the level 2 module, RNGSND, to the PNCDIM. This queue holds buffers of data that are to be transmitted by the PNC. If this count indicates that most of the buffers are being held in this queue, then there is congestion on the ring and the buffers are not being transmitted quickly enough. This could happen when there were several nodes WACKing heavily.

Receive Queue Full: This queue is allocated to be large enough to hold all the buffers simultaneously. It is, therefore, an error for this queue to run out of room. If this situation does occur, the buffer being queued is discarded and an event logged.

WACKed Packets: This is the same counter that was shown on the Basic Display.

WACK List Entry Count: If a packet is WACKed 20 times in a row, it is placed on the WACK Linked List. If another packet for the same node is encountered, it is placed on this list behind the first. The list is held in node number sequence. This is a cumulative counter that shows how many packets were placed on the list since the system was cold started or the counters were zeroed. It is not an indication of the current length of the list.

WACK List Max Entry: This counter will tell the maximum number of packets on the list at one time since cold start.

The WACK List fields depict the pattern of WACKing on the ring. If a single node is sending out WACKS frequently, usually only one or two packets will be on the list at one time. If there are bursts of WACKS on the ring, then this count will be much higher.

Fatal Errors: These errors concern hardware I/O operations, and rarely occur. If a fatal error occurs, RINGNET removes the node from the ring. The ring will be marked as disabled, the PNC will be told to disconnect (if possible), and all transmit packets will be aborted. The PNC diagnostics should be run before any attempt is made to restart the node on the ring. Fatal errors are logged as events in NET_LOG.

Trace Display

The Trace Display shows details of the traffic between the local node and all other nodes on the ring. It assists you in monitoring traffic flows and in investigating problems between nodes. The Trace Display, shown in Figure 23-3, contains data only if the -TRACE_NODE option was specified in the START_NET command (refer to Chapter 17 STARTING UP PRIMENET). Only nodes that have had activity appear on this display. The Trace Display is intended to isolate individual node activity on the ring. It only tracks data sent by or received by the node on which MONITOR_RING is being run. The data are a subset of those that appear in the Basic Display.

```
Trace Display  ((commands:  H for help,  T restarts trace ))
               ((           RETURN for next trace section ))

Node Name  Node ID    Transmitted    Received     WACKed
cccccc        nnn          xxxxx       xxxxx      xxxxx
cccccc        nnn          xxxxx       xxxxx      xxxxx
cccccc        nnn          xxxxx       xxxxx      xxxxx
cccccc        nnn          xxxxx       xxxxx      xxxxx
cccccc        nnn          xxxxx       xxxxx      xxxxx
cccccc        nnn          xxxxx       xxxxx      xxxxx
```

Trace Display
Figure 23-3

The Trace display heading includes an identifier and a reminder of some single-key commands. You use T to start the initial Trace display; you press the RETURN key to advance the display. The five columns making up the Trace display are described below.

Node Name, Node ID: This column shows the name and the identification number of the node to which the local node was communicating.

Transmitted: This column shows the number of packets correctly transmitted to that node. This total does not include any broadcast messages.

Received: This column shows the number of packets correctly received from that node, including any broadcast messages sent by that node.

WACKed: This column shows the number of packets the local node attempted to send that were Wait Acknowledged by the other node.

Report File Format

MONITOR_RING archives data to a file in a fixed format. This declaration allows analysis by user-written software. PRIME reserves the right to modify the contents and layout of this file. Forward compatibility is not guaranteed. This declaration appears in the MONITOR_RING source code.

The following is a Pl/1 description of the format of data contained in a REPORT file. If you do not select the TRACE option, then the trace section at the end of the data block will not be included in the file specified with the REPORT option.

```
dcl 1 iobuf,
    2 record_idfr,
    3 size fixed bin,                  /* Size, in words, of this data block */
    3 version fixed bin,               /* Version of MONITOR_RING  */
    2 sample_time char(8),             /* Time of sample: 'hr:mn:sc' */
    2 sample_date char(16),            /* Date: 'Tues, Jun 24, 1983'  */
    2 sample,
    3 packets_received fixed bin(31), /* Number of packets received  */
    3 words_received fixed bin(31),    /* Number of WORDs received  */
    3 rcverrcnt fixed bin(15),         /* Number of receive errors  */
    3 rcverr fixed bin(15),            /* Receive error status  */
    3 rcverrnode fixed bin(15),        /* Node originating erring packet  */
    3 packets_xmited fixed bin(31),    /* Number of packets transmitted  */
    3 words_xmited fixed bin(31),      /* Number of WORDs transmitted  */
    3 wacks fixed bin(31),             /* Number of Wait ACKnowledgments  */
    3 xmtnoackcnt fixed bin(31), /* Number of non acknowledged packets  */
    3 multt fixed bin(15),         /* Number of times multiple tokens seen  */
    3 multa fixed bin(15),         /* Number of multiply acceptance pkts */
    3 xmtstuck fixed bin(15),      /* Number of non-returned transmits  */
    3 xtra_xint fixed bin(15),     /* Number of spurious transmit interrupts */
    3 xmterrcnt fixed bin(15),     /* Number of transmit errors  */
    3 error_status fixed bin(15),/* Status of transmit error  */
    3 xmtfailcnt fixed bin(15),    /* Number of aborted transmits  */
    3 tokens fixed bin(15),       /* Number of tokens inserted into the ring */
    3 wacknode fixed bin(15),     /* Node number of node that WACKed out  */
    3 dupnode fixed bin(15),      /* Duplicate node id  */
    3 nobuf_count fixed bin(15),/* Number of times no rcv buffer available */
    3 flto2_count fixed bin(15),/* Number of buffers q'd to protocol layer */
    3 f2tol_count fixed bin(15),/* Number of buffers queued to DIM  */
    3 xmit_bdcast fixed bin(31),/* Number of broadcast messages sent  */
    3 rcv_bdcast fixed bin(31), /* Number of broadcast messages received  */
    3 cntr_ina fixed bin(15),     /* Controller blew up on INA  */
    3 cntr_rcv fixed bin(15),     /* Controller not acptng receive OTA  */
    3 cntr_xmt fixed bin(15),     /* Controller not acptng transmit OTA  */
    3 cntr_dma  fixed bin(15),    /* Controller DMA failure  */
    3 int_rcv fixed bin(15),      /* Spurious receive interrupt  */
    3 res_fail fixed bin(15),     /* flto2 q full  */
    3 wack_cnt fixed bin(15),     /* Number of packets put on Wack List  */
    3 wack_max fixed bin(15),     /* Maximum on list at one time  */
    3 badprot fixed bin(15),      /* Received packets with bad protocol field */
    3 my_node fixed bin(15),      /* Node id of this node  */
    3 my_node_name char(6),       /* Name of this node  */
    3 rcv_node_name char(6),      /* Name of node causing receive error  */
    3 xmt_node_name char(6),      /* Name of node causing transmit error  */
    3 dupnode_name char(6),       /* Name of duplicate node  */
    2 tracebuf (248),             /* Buffer for tracing traffic info  */
    3 node_id fixed bin,          /* Node-id re: this traffic info  */
    3 node_name char(6),          /* Node name re: this traffic info  */
    3 rcvpkt fixed bin,           /* Number of pkts received from other node  */
    3 xmtpkt fixed bin,           /* Number of pkts transmitted to other node */
    3 wakpkt fixed bin;           /* Number of pkts WACKed by other node  */
```

Note

All character data are Prime ASCII. All numeric fields are full 16/32 bit UNSIGNED integers, not 15/31 bit signed integers.

THE FIND_RING_BREAK PROGRAM

You use the FIND_RING_BREAK program to locate a break in the ring. This program works only for 'hard' breaks, which cause complete interruption of the signals on the ring. The operator runs FIND_RING_BREAK when the following happens.

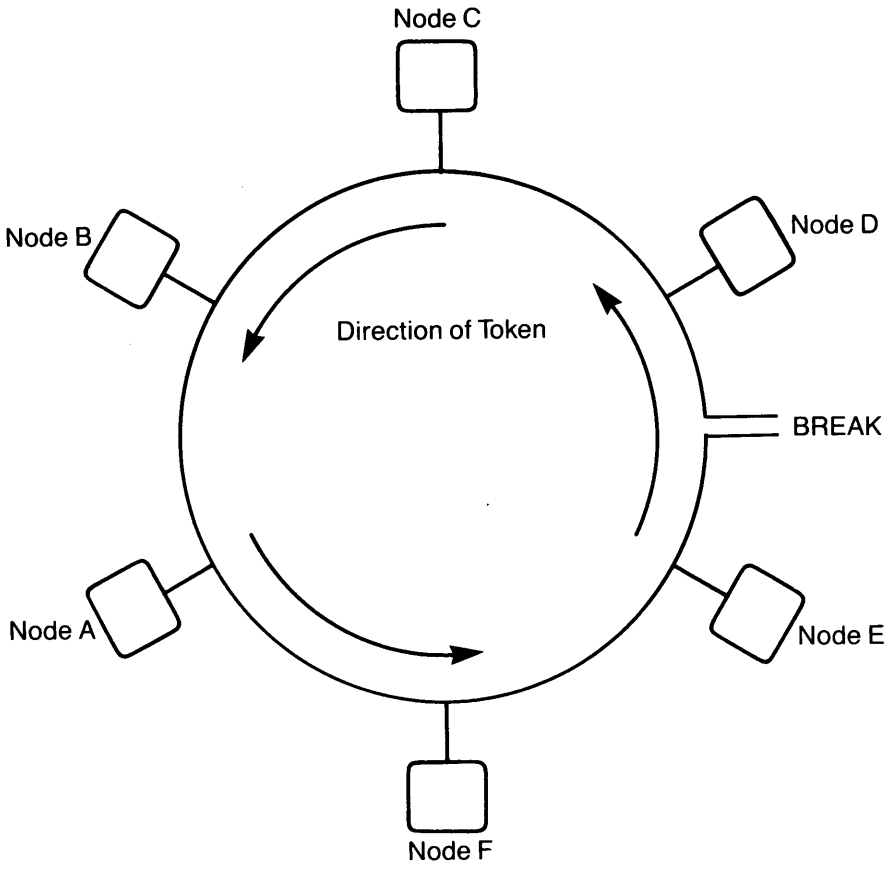
- The MONITOR_RING program indicates a break
- The "RING MAY BE DOWN" error message appears on the supervisor terminal
- The STAT NET command indicates "down" nodes

The following sections describe:

- How to invoke FIND_RING_BREAK
- How FIND_RING_BREAK works
- What the input file is and how to create it
- How to locate a break in the ring
- FIND_RING_BREAK error messages

How FIND_RING_BREAK Works

To determine the location of a break in the ring, FIND_RING_BREAK analyzes the local node's ability to communicate with the other nodes in the ring. Specifically, the local node will be able to hear from most active nodes downstream of any break, yet upstream from the local node. An example is shown in Figure 23-4: there are six nodes along a ring, configured to communicate with each other. Node A is the node on which FIND_RING_BREAK is being run. There is a break somewhere between Node E and Node D.



Locating a Break in a Ring
Figure 23-4

When FIND_RING_BREAK is run on Node A, the following occurs. Node A recognizes that it is hearing from, but can not send to, Nodes B and C. The reason that Node A can determine that Nodes B and C are active is because Node D is still transmitting a token. Node B and Node C are using this token to transmit. Node C, the first node downstream of the first active node after the break (Node D) can use that token to transmit data. Node D cannot transmit data because no node upstream of Node D is sending out a token; it can only issue a token onto the ring.

Once FIND_RING_BREAK determines the node farthest upstream from which it is receiving data (Node C), it then looks upstream for the next configured node (Node D). It knows that Node D must be active for Node C to transmit data. The program then identifies the break as being somewhere between Node D and the next active node. In this case, the break in the ring is at or after Node E.

Note

The node issuing a token may not be seen as active if it is running a version of the software earlier than Rev. 19.3 and is not configured to talk to the node on which FIND_RING_BREAK is being run. Therefore, the inactive nodes will have to be taken into consideration.

Normally, a problem with a connector in the ring is to blame. But, occasionally, a defective PNC board or ring cable may be the cause.

Invoking FIND_RING_BREAK

The syntax for FIND_RING_BREAK and an explanation of the options are shown below.

```
FIND_RING_BREAK [-HELP
                -INPUT filename]
```

-HELP -H	Displays the syntax and describes the options of this command.
-INPUT filename -I	The name of a file that contains the physical configuration of the ring. The default is no file.

Before invoking FIND_RING_BREAK, you may want to create an input file that describes the physical configuration of the ring.

The FIND_RING_BREAK Input File

Because ring nodes are not necessarily located in node id sequence or in the sequence in which they are entered into the configurator, it is useful to tell FIND_RING_BREAK what the actual physical configuration of the ring is. You put this information into a file that FIND_RING_BREAK uses to specify the location of a break.

When you run FIND_RING_BREAK using an input file, FIND_RING_BREAK displays all the nodes specified in the file in the configuration sequence, beginning with the local node and going downstream. Down nodes, active nodes and inactive nodes are listed. After this list, FIND_RING_BREAK prints the names of three nodes and an interpretation of where the break is.

FIND_RING_BREAK assumes that the node names in the input file are correct; it does not check them.

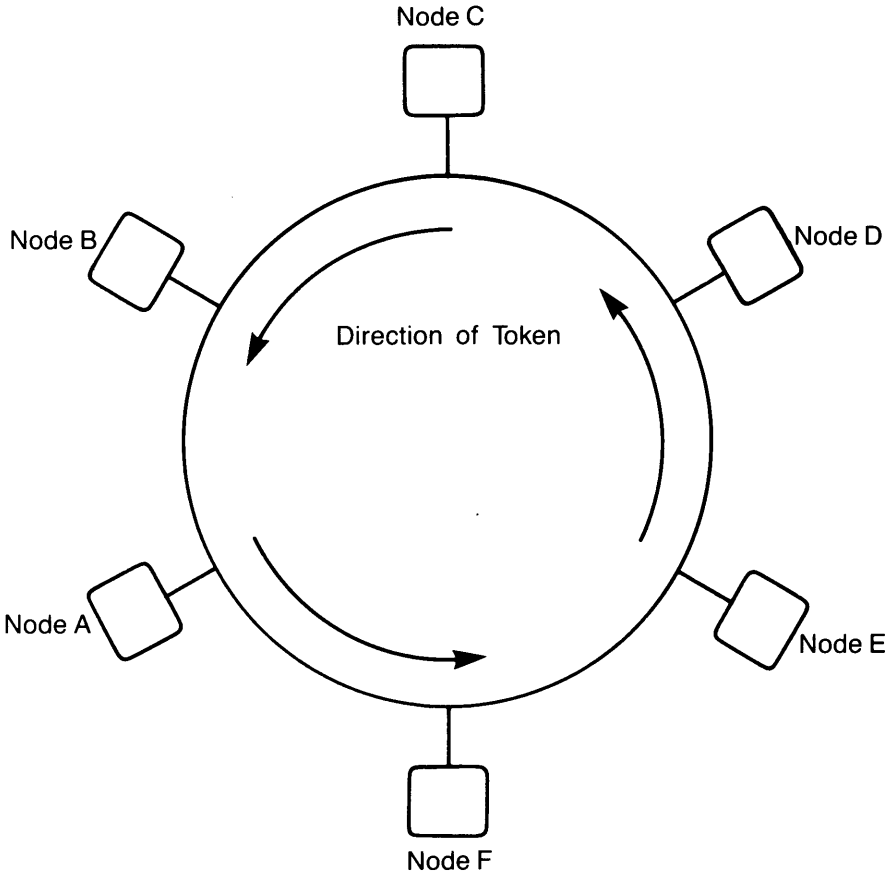
When run without an input file, FIND_RING_BREAK does not attempt to interpret the location of the break. It only prints the names of those nodes in the network configuration file, and its display shows only active nodes. The chart below compares using FIND_RING_BREAK with and without an input file.

Feature	File	No File
Shows actual configuration	yes	no
Shows all configured nodes	yes (if in file)	yes
Lists inactive nodes	yes	no
Displays node names	yes	configured nodes only
Displays node ids	yes	yes
Interprets location of break	yes	no
Lists active nodes	yes	yes
Lists down nodes	yes	yes

Creating the FIND_RING_BREAK Input File

The FIND_RING_BREAK input file describes the physical arrangement of the nodes on the ring. You create this file with any editor. The file may begin with any node. From that point on, you list the remaining nodes in the order in which the token travels around around the ring from that first node (counterclockwise).

There should be only one entry or line per node. On each line put the node-id and the node name. For example, let us use the ring shown in Figure 23-5 to construct a sample file.



Sample Ring
Figure 23-5

We begin the file with the node id and node name of Node A. We then enter the remaining nodes in the order in which data travel around the ring: Node F, Node E, Node D, Node C, and Node B. The example below shows how the file would look.

```
001 A
006 F
005 E
002 D
004 C
003 B
```

Node ids must be three digits. Leading zeroes are required. Node names can be up to six characters. There must be a space between the node number and name.

LOCATING A BREAK IN THE RING

The next two sections describe how to locate a break in the ring by using FIND_RING_BREAK with an input file and without one.

FIND_RING_BREAK With an Input File

If you invoke FIND_RING_BREAK with the -INPUT option, the program attempts to locate and read the specified input file. If FIND_RING_BREAK is unable to read the specified input file, you will be asked:

```
Shall we continue without a file? [YES/NO] NO
```

If you answer NO, the program ends. If you answer YES, FIND_RING_BREAK uses the information in the network configuration and proceeds as if no file was specified. Refer to the next section, FIND_RING_BREAK Without an Input File.

If FIND_RING_BREAK can read the input file, it asks you:

```
Do You Want To Reset the Information? [YES/NO]
```

Normally, you answer NO to this prompt. If you suspect a break exists and FIND_RING_BREAK does not indicate it, you should wait several minutes before you re-invoke the program. Reset the database only if repeated running of FIND_RING_BREAK shows that the network is up when MONITOR_RING or ring operation indicates that the ring is down. If you answer YES, FIND_RING_BREAK displays the current information, and then resets the database.

If you answer NO or press the RETURN key, FIND_RING_BREAK accesses the database, which contains the data on the current condition of the ring.

Note

If FIND_RING_BREAK successfully locates your node-id, it then prints out the following display. (nnn) indicates a 3-digit node-id. XXXXXX represents a 6-digit node name.

```
OK, FIND_RING_BREAK
Do You Want To Reset the Information? [YES/NO] NO
LIST OF NODES IN CONFIGURATION SEQUENCE
(nnn) XXXXXX - DOWN
(nnn) XXXXXX - INACTIVE
(nnn) XXXXXX - DOWN
(nnn) XXXXXX - DOWN
(nnn) XXXXXX - UP
*****************
(nnn) XXXXXX
(nnn) XXXXXX
****** BREAK ******
(nnn) XXXXXX
Retry? [Y/N] N
```

Each time the program has printed 20 lines, it will stop and print.

```
More - hit cr
```

FIND_RING_BREAK Without an Input File

If you have not specified an input file with the -INPUT option, FIND_RING_BREAK uses the information in the network configuration.

```
OK, FIND_RING_BREAK
Do You Want to Reset the Information? [YES/NO] YES
List of Active Nodes in Node Id Sequence
(nnn) - UP
(nnn) - UP
(nnn) - DOWN
(nnn) - UP
Retry? [Y/N] N
```

When you do not use an input file, FIND_RING_BREAK cannot attempt to specify the location of the break.

FIND_RING_BREAK ERROR MESSAGES

There are two error messages that can occur when FIND_RING_BREAK attempts to read the database. These errors cause FIND_RING_BREAK to discontinue the program and return to PRIMOS level.

```
Ring Not Configured
```

The local node is not aware of the RINGNET.

```
Node Not In Ring
```

The local node is disconnected from the ring.

APPENDIXES

A

X.25 Programming Guidelines

INTRODUCTION

This appendix contains information about X.25 protocol as it relates to programming with IPCF subroutines. The following topics are described.

- Call user data field terminology
- X.25 window and packet size
- Data flow checkpoints

CALL USER DATA FIELD TERMINOLOGY

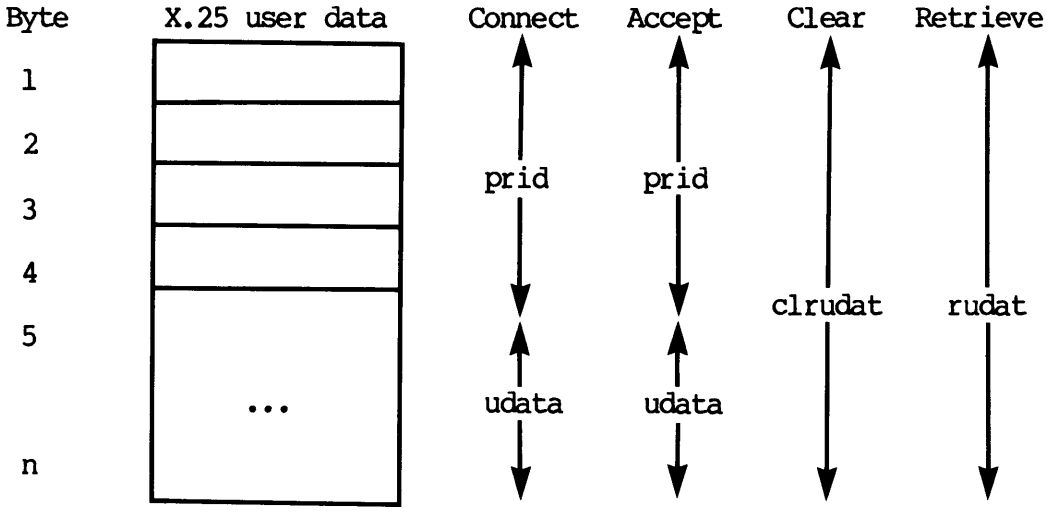
X.25 permits an application to place a call user data field in its call request packet. The maximum size of the user data field is 16 bytes, except for fast select call requests, which may be up to 128 bytes long. Also, a call accept packet responding to a fast select call can contain a called user data field, and similarly, a clear request packet responding to a fast select call can contain a clear user data field.

The CCITT standards X.3, X.28, and X.29 regulate PDN protocol. These standards use the first four bytes of the call user data field as a protocol identifier field. The remainder of the call user data can be used for call data.

X.25 acknowledges the existence of X.3 and other protocols by setting rules for the two most significant bits of the first byte of the call user data and the called user data. For applications, these two bits should both be set to 1. However, there is no restriction on these bits in clear user data.

The IPCF routine arguments recognize the protocol identifier field by handling it as a separate argument for connect and accept 'long-form' routines. PRIMENET uses the full 4-byte protocol identifier field to convey the port number information in call requests. Thus, this 4-byte field for Prime-to-Prime virtual circuits cannot be used in applications. In contrast, the protocol identifier argument is not used as part of clear user data for the fast-select form X$FCLR.

When initiating a call request, an application can provide an array to retrieve any returned user data. This array corresponds to the X.25 user data field, and thus includes the protocol identifier field. The picture below shows how the call/called/clear user data, as defined by X.25, are split on IPCF arguments. It also shows the retrieve-data array, optionally provided by the connect call.



X.25 WINDOW AND PACKET SIZE

The following table defines X.25 international facility formats for determining window and packet sizes. The default values, provided by PDNs, are a window size of 2 and a packet size of 128 bytes. The window and packet size facility elements consist of 3 bytes each. You can include either or both in the facility field, and combine them with other facility elements.

Facility element	First Byte	Second Byte	Third Byte
Window size	01000011	00000xxx	00000yyy
Packet size	01000010	0000zzzz	0000vvvv

Parameter	Meaning
xxx	Window size, binary coded, (000 not allowed) for transmission called node to calling node
yyy	Window size, binary coded, (000 not allowed) for transmission calling node to called node
zzzz	Packet size code (see below) for transmission called node to calling node
vvvv	Packet size code (see below) for transmission calling node to called node

The packet size code is the binary coded logarithm base 2 of the packet size, expressed in bytes.

zzzz/vvvv	0100	0101	0110	0111	1000	1001	1010
Size (bytes)	16	32	64	128	256	512	1024

The maximum supported packet size is configurable to 512, 1024, or 2048 for RINGNET (see the Network Planning and Administration Guide), and is 256 bytes for synchronous lines.

The description above limits the window size to less than or equal to 7, which is the maximum permitted for normal X.25 sequence numbering, and also the maximum value currently supported by PRIMENET. (X.25 as such also defines extended sequence numbering which will permit larger window sizes.) For example, suppose you wanted for the calling node's transmissions a window size of 7 and a packet size of 256, and for the called node's transmissions the normal default values of 2 and 128. The facility field required would be the following six bytes:

01000011 00000010 00000110 01000010 00000111 00001000

DATA FLOW CHECKPOINTS

The aim of X.25 levels 2 and 3 is to provide error-free data transmission with ordered flow. That is, data delivery at the receiver takes place transparently and in the order in which it was delivered by the sender. However, there is always a slight possibility that the error handling routines of levels 2 and 3 might fail; your virtual circuit is reset. The reset means that some packets may have been

lost, although you do not know how many packets or in what transmission direction.

If a reset occurs, any buffer given to X$RCV, but which has not yet been given back with XS$CMP, must be regarded as an error. Also, X$TRAN calls interpret XS$CMP to mean that the buffer has been successfully enqueued locally for transmission. However, the buffer may still have been lost in transmission. You might consider using a checkpoint system to check for resets and other errors. That is, at certain intervals in the data stream, you should define and transmit/acknowledge checkpoint messages. If a reset or other error occurs, you can then require retransmission from the most recent checkpoint. Breaking a large file down into smaller components outside of the IPCF application is also a checkpoint system.

The error rate for RINGNET is considerably lower than the usual rate for synchronous (long distance) lines. It is not practical to use checkpoints for a RINGNET-only application. However, the time loss from repeated retransmission of a large file over a synchronous line could justify the use of checkpoints.

B
NETLINK Parameters

This appendix presents the X.3 (level 1) terminal characteristics parameters supported by NETLINK. These parameters are decimal values that are set with NETLINK's SET command and displayed with NETLINK's PAR command.

NETLINK supports only the TELENET and DATAPAC national options. Supported TELENET parameters are shown here. See appropriate TELENET literature for additional information about use of these parameters. See DATAPAC literature for information on the DATAPAC parameters. See Consultative Committee for International Telephone and Telegraph (CCITT) literature for more information.

INTERNATIONAL PARAMETERS

0 - National options marker

(Only the values 0 for DATAPAC Canada, and 33 for TELENET USA, are currently supported.)

2 - Echo (0 for no echo, 1 for echo)

3 - Data forwarding

0 - No data forwarding character
1 - Alphanumeric characters (A-Z, a-z, 0-9)
2 - Carriage return
4 - ESC, BEL, ENQ, ACK
8 - DEL, CAN, DC2
16 - ETX, EOT
32 - HT, LF, VT, FF
64 - All other control characters

The only valid combinations are
0, 2, 6 (2+4), 18 (2+16), 126 (2+4+8+16+32+64)

4 - Idle timer Delay (number of 50 ms increments, any number from 0 to 255; 0 for none)

7 - Break handling

0 - Nothing
1 - Interrupt
2 - Reset
4 - Send indication of break
8 - Escape to command mode
16 - Discard output

The only valid combinations are 0, 1, 2, 8, 21 (1+4+16)

8 - Discard output

Used when parameter 7 is 21. 0 indicates normal data transmission, 1 indicates output is to be flushed. Not to be set by you.

12 - DTE to DCE Flow Control (1 = use XOFF and XON, 0 = do not)

TELENET PARAMETERS

NETLINK supports the CCITT parameters 1 through 18. It also supports DATAPAC (national options 0) parameter 126, which is Linefeed Insertion, and the following TELENET (national options 33) parameters:

1 Linefeed Insertion
2 Network Message Delay
3 Enable Local (network) Echo
5 Data Forwarding Characters
9 CR padding
10 LF padding
12 Line Width
13 Page length
16 Interrupt on Break (TELENET discontinued)
17 Break Code (TELENET discontinued)

18 NVT options
27 Delete character
28 Cancel character
29 (Line Re-)Display character
30 Abort output character (TELENET discontinued)
34 Transmit on Timers
35 Idle Timer
36 Interval Timer
40 Insert code on BREAK (TELENET discontinued)
45 Send APP on BREAK (TELENET discontinued)
54 DTE-to-DCE flow control (TELENET discontinued)
58 Connection Escape Enable
59 Flush on BREAK (TELENET discontinued)

The default parameters for X.25 are

- Reverse charging
- X.29 PAD
- Unknown terminal type
- Escape character @
- Inhibit remote echo
- Polling rate of 1/20 second
- Terminal speed of 1200 baud

INTERPRETING PAR COMMAND OUTPUT

The following list shows the relationship between the messages displayed by the PAR command with no options, and the X.3 parameter values:

PAR Output	Param.	Value
FULL DUPLEX	2	1
FORWARD DATA ON	3	
Alphanumerics		1
CR		2
ESC		4
Editing		8
Terminators		16
Form		32
Other Cntrl		64
Other Printing		128
IDLE TIMER = x.x Secs.	4	Units of 50 ms

ON BREAK	7	(Legal values: 0, 1, 2, 8, 21)
Interrupt		1
Reset		2
Send indication of break		4
Escape to command mode		8
Discard output		16
Send APP	TELENET break handling	
Insert	TELENET break handling	
OUTPUT BEING DISCARDED	8	1
X-OFF/X-ON ENABLED	12	1
Local Editing Enabled	15	1
NVT Process Control Enabled	(TELENET Network Virtual Terminal)	

C

FTS Error Messages

INTRODUCTION

This appendix lists and explains each FTS error that occurs in the FTGEN, FTR, and FTOP utilities, and in applications that use the FT$SUB subroutine. The FTGEN utility, which initializes servers, queues, and sites, is described in the Network Planning and Administration Guide. FTR, FTOP, and the FT$SUB subroutine are described in this book.

GENERAL ERROR MESSAGES

The following errors can occur in any of the FTS utilities.

- Argument too long. (F$ARTL)

You specified an argument that was longer than the maximum allowed argument length.

- FTS not ready for use. (Q$QNRD)

The FTS data base has not been initialized with the FTGEN INITIALIZE_FTS command.

- The FTS data base is invalid. (Q$NVDB)

The FTS data base has been corrupted, or an FTGEN INITIALIZE_FTS command has not been performed after FTS installation.

FTGEN ERROR MESSAGES

The following error messages may occur when you are using FTGEN.

- Address of site invalid. (F$ADIN)

The site address you specified is invalid. The address must be less than 128 characters long and contain a plus sign (+) character that delimits the address.

- Argument is not numeric. (F$ARNN)

You specified a nonnumeric argument. You must specify a number in the argument.

- Bad site name format. (F$BDSN)

This message occurs in FTGEN as well as FTR. The site name must adhere to the Prime filename standard. See the Prime User's Guide for more information on naming standards.

- Cannot modify this parameter. (F$CMDP)

You tried to modify the maximum number of file transfer requests for a particular queue, which is not allowed. The maximum number of requests can only be specified when adding the queue, not modifying it.

- Invalid message level. (F$INMS)

You specified an FTS log file message level other than the following valid ones: NORMAL (1), DETAILED (2), STATISTICS (3), and TRACE (4). This message occurs in FTR as well as in FTGEN.

- Invalid range. (F$IRNG)

The range you specified is numerically invalid for the particular option. For example, if you add a server and set PORT to -3, an invalid number, this error is displayed. Valid numbers are between 1 and 99 for a port.

- No queues configured. (Q$NOQC)

You attempted to list all the queues, but none were configured. Ask your System Administrator about your system's current FTS configuration.

- Queue does not exist. (Q$QNEX)

The specified queue has not been configured with FTGEN. This error occurs in FTR as well as in FTGEN.

- Queue is already associated with another server. (F$QAAS)

The queue you specified is already in use, so you must specify another queue name in the command.

- Queue name invalid. (F$QNIN)

The queue name you specified is invalid. The queue name must conform to Prime filename syntax and cannot contain any full stop characters. See the Prime User's Guide for information on standard naming syntax.

- Queue in use. (F$QUIU)

You attempted to delete an empty queue, but the server associated with the queue still exists.

- Queue not empty. (F$QUNE)

You attempted to delete a queue that has requests in it. Wait until the queue is empty to delete it.

- Text follows last argument. (F$TLFA)

This error occurs in FTOP as well as in FTGEN. The argument to a particular command contains extra text. For example, if you specify a server port with the following command, the number 14 is acceptable, but FRED is an incorrect argument:

```
PORT 14 FRED
```

- Too many servers. (F$TMSV)

You tried to configure a server and exceeded the maximum number of servers. (Eight is the maximum.)

- Too many queues. (F$TOMQ)

You tried to configure a queue and exceeded the maximum number of queues. (Eight is the maximum.)

- Unknown command. (F$UNCD)

You did not type an FTGEN command. See the Network Planning and Administration Guide for a list of FTGEN commands.

- You cannot delete a server while it is running. (F$CDLS)

You tried to delete a server while it was still running. Use the FTOP -STOP_SRVR command to stop the server.

- You cannot modify a queue while its server is running. (F$CMDQ)

You tried to modify the queue while its associated server is running. You can only modify a queue when its associated server is not running. Use the FTOP -STOP_SRVR command to stop the server.

- You cannot modify a server while it is running. (F$CMDS)

You tried to modify an active server. You can only modify a server when it is not running. Use the FTOP -STOP_SRVR command to stop the server.

- You do not have operator privileges. (F$NOPP)

You attempted to invoke the FTGEN command without being logged in as SYSTEM. You must log in as SYSTEM to use the FTGEN command.

FTR ERROR MESSAGES

The following messages occur in the FTR utility, which allows users to submit and manage file transfer requests.

- Bad command line format. (F$BDCL)

The format of the command line is incorrect. For example, you should type FTR LETTER -CANCEL instead of FTR -CANCEL LETTER.

- Bad device name. (F$BDDN)

You typed an invalid -DEVICE name. LP is the only correct device name.

- Bad site name format. (F$BDSN)

This message occurs in FTGEN as well as in FTR. The site name must be a valid one and adhere to the Prime filename standard. See the Prime User's Guide for more information on naming standards.

- Conflicting options. (F$CNOP)

The options you specified conflict. For example, you specified a request with both -COPY and -NO_COPY when it must be either one or the other option.

- Copy option only applies to local source file. (F$CPLS)

You specified this option when you were fetching a file from a remote site, which is not allowed. This option is relevant only when you are sending local files.

- Copy flag may not be modified. (F$CPMD)

You tried to modify the copy flag, which is not allowed. For example, the following command would get this error:

```
FTR -MODIFY 2 -COPY
```

- Destination file may not be modified. (F$DFMD)

You tried to modify the destination file option, which is not allowed.

- Destination file has not been specified. (F$DFNS)

You did not specify a destination pathname. See Chapter 5 for the format of an FTR command line.

- Delete option only applies to local source file. (F$DLLS)

You specified this option when you were fetching a file from a remote site, which is not allowed. This option is relevant only when sending local files.

- Device transfer from remote site not allowed. (F$DRNA)

You attempted to fetch a file from a remote site and send it a local device, which is not allowed.

- Destination site may not be modified. (F$DSMD)

You tried to modify the destination site name, which is not allowed.

- Destination site is not configured. (F$DSNC)

The destination site has not been configured in the FTS configuration for your site.

- Destination user name invalid. (F$DUIN)

You used an invalid destination user-id. The user-id must conform to Prime's standard for user-ids. See the Prime User's Guide for more information.

- Destination user not specified when destination notify requested. (F$DUNS)

You did not specify a destination user (with -DSTN_USER). You must specify a destination user if you use the FTR -DSTN_NTFY option.

- Duplicate option. (F$DUOP)

You duplicated one or more options. For example:

```
FTR <ASH>TREE <ELM>BRANCH -SRC_NTFY -DSTN_USER CLARKE -SRC_NTFY
```

duplicates the -SRC_NTFY option, which is not allowed.

- Full pathname too long. (F$FPTL)

You exceeded the maximum pathname length of 128 characters.

- Hold flag may not be modified. (F$HDMD)

You tried to modify the hold flag, which is not allowed. For example, the following command would produce this error:

```
FTR -MODIFY 3 -HOLD
```

- Invalid destination file type. (F$IDFT)

You did not specify a correct destination file type. See the -DSTN_FILE_TYPE option description in Chapter 6 for more information.

- Invalid external name. (F$INEX)

The name you specified for the request name is not a valid Prime file name. See the Prime User's Guide for the correct filename syntax.

- Illegal file or directory conversion. (F$IFDC)

You used the -SRC_FILE_TYPE or -DSTN_FILE_TYPE options in an invalid combination. See Chapter 6 for more information on these options.

- Invalid message level. (F$INMS)

You specified an FTS log file message level other than the following valid ones: NORMAL (1), DETAILED (2), STATISTICS (3), and TRACE (4). This message occurs in FTGEN as well as in FTR.

- Invalid source file type. (F$ISFT)

You specified an incorrect source file type. See Chapter 6 for more information on the -SRC_FILE_TYPE option.

- Message level specified but request log treename omitted. (F$MBNL)

You specified the -MSGL_LEVEL option with a specific level, for example, DETAILED, but you did not specify a log filename.

- Missing command line parameter. (F$MCLP)

The command line has a required parameter missing. For example:

```
FTR <ELM>TREE <ASH>BURN -SRC_NTFY -DSTN_USER
```

did not specify a destination user-id with the -DSTN_USER option.

- Networks unavailable.

You tried to use FTR to submit a request, but the network has been shut down or not configured.

- No copy option only applies to local source file. (F$NCLS)

You specified this option when you were fetching a file from a remote site, which is not allowed. This option is relevant only when you send local files.

- No Copy flag may not be modified. (F$NCMD)

You tried to modify the NO_COPY flag, which is not allowed. For example, the following command would cause this error:

```
FTR -MODIFY 2 -NO_COPY
```

- No delete option only applies to local source file. (F$NDLS)

You specified this option when you were fetching a file from a remote site, which is not allowed. This option is relevant only when you are sending local files.

- No eligible request of this name found. (F$NERF)

You attempted to modify, abort, release, hold or cancel requests with the specified name without success because either they do not exist or they are in an ineligible state. For example, you would receive this error if you tried to hold a request that was already HELD.

- No request of this name found. (F$NRFD)

You specified a nonexistent request name when you performed a -DISPLAY or -STATUS of a particular request. Check that you specified the right name, or use the request number in the command.

- No requests queued. (F$NRQD)

You tried to list the contents of a queue that is empty.

- Not configured. (F$NTCF)

You specified a site, server, or queue that had not been configured with FTGEN.

- Only one management option allowed. (F$OMOP)

You specified more than one management option, which is not allowed. For example, FTR -ABORT LETTER -CANCEL is not allowed.

- Passworded pathname must be fully qualified. (F$PSFQ)

You did not specify a passworded pathname from the UFD down to the filename, and you did not enclose the complete pathname, including the password, in single quotes.

- Queue blocked. (Q$QBLK)

You tried to submit a request to a queue that has been blocked with the FTGEN BLOCK_QUEUE command. The queue must be unblocked with the FTGEN UNBLOCK_QUEUE command so that requests can be accepted.

- Queue does not exist. (Q$QNEX)

You tried to submit a request to a request queue that has not been configured with FTGEN.

- Queue full. (Q$FULL)

The request queue is full.

- Queue name may not be modified. (F$QNMD)

You tried to modify the -QUEUE option, which is not allowed.

- Request held by operator. (F$RHPR)

You tried to release an operator-held request.

- Request log treename same as source or target treename !! (F$RLST)

You specified a log filename that is not different from the source or destination pathname.

- Request already aborting. (F$RQAB)

You tried to use an FTR management option (except -STATUS or -DISPLAY) on an aborting request, which is not allowed.

- Request already put on hold by FTS. (F$RQHF)

You tried to hold or abort a request that has already been held by FTS, which is not allowed.

- Request already put on hold by operator. (F$RQHO)

You tried to hold or abort a request that has already been held by an operator, which is not allowed.

- Request already put on hold by user. (F$RQHU)

You tried to hold or abort a request that has been put on hold already.

- Request waiting. (F$RQWT)

You tried to release or abort a waiting request, which is not allowed.

- Remote treename incorrectly specified. (F$RTIS)

You specified a pathname for the destination site that did not include disk and directory names. You must specify the entire pathname in the command line.

- Segment dir. transfer to/from a Rev 1 site is not supported. (F$PINS)

You tried to transfer a SEG file to or from a REV 1 FTS site, which is not allowed.

- Source or destination site must be local. (F$SDSL)

You cannot make file transfers between two remote sites. You can transfer requests in loopback on a local site, or between local and remote sites only.

- Source file type may not be modified. (F$SFMD)

You tried to modify the source file type, which is not allowed.

- Source file does not exist. (F$SFNE)

You tried to transfer a nonexistent file. Check to see that you specified an existing file for the transfer request.

- Source file has not been specified. (F$SFNS)

You did not specify a file to be sent or fetched in the transfer request.

- Specified and actual source file types differ. (F$SFTD)

You used the -SRC_FILE_TYPE option, but the file type that you specified differs from the actual source file type.

- Source site may not be modified. (F$SSMD)

You tried to modify the source site, which is not allowed.

- Source site is not configured. (F$SSNC)

You specified a source site that has not been configured with FTGEN.

- Source user name invalid. (F$SUIN)

You specified an incorrect source user-id. User-ids must conform to Prime's naming standard. See the Prime User's Guide for more information.

- Source user not specified when source notify requested. (F$SUNS)

You did not specify the source user (with -SRC_USER). You must specify a source user if you use the -SRC_NTFY option in the command line.

- Transfer to a device as well as a file is not allowed. (F$TDFN)

You cannot specify both a destination file and a destination device (-DEVICE LP) in one transfer request.

- Transferring a SEG directory to a DEVICE is not supported. (F$TDNS)

You cannot print a SEG file type on a remote line printer.

- Transferring a file to itself is not possible. (F$TFNP)

You used only one filename for two files in a transfer request. The source and destination file cannot be the same.

- Transfer in progress. (F$TRPR)

You tried to release, cancel, modify, or hold a transferring request, which is not allowed.

- Unable to create temporary file. (Q$UCTF)

The number of temporary files in the FTSQ* directory has reached the maximum number as a result of queued requests. The operator should investigate the possibility of any old requests being canceled.

- Unknown keyword. (F$BDKW)

An argument on the command line is unknown. For example:

```
FTR <ASH>TREE <ELM>BRANCH -FRED
```

shows an unknown keyword, -FRED. The command line should also include -DSTN_USER (abbreviation is -DS) for the user-id FRED.

- Unknown option. (F$UNOP)

You specified an unknown option in the command line. This error occurs in FTOP as well as in FTR if an extra option is specified. For example:

```
FTR <ELM>TEST <ASH>ANSWERS -DSTN_USER JONES -SRC_NTFY EXTRA
```

specifies an extra option, -EXTRA, that is not known to FTS.

FTOP ERROR MESSAGES

The following section describes the errors that occur in the FTOP utility, which is the FTS operator utility for starting and stopping FTS servers and the FTS manager (YTSMAN).

- FTS manager already notified to close down. (F$MNAC)

You tried to stop the FTS manager (YTSMAN) with the -STOP_MNGR command, but it has already been notified to stop with a previous FTOP -STOP_MNGR command.

- Server link is not active. (F$SLNA)

You tried to abort a server link that is not active. Use the FTOP -LIST_SRVR_STS command to see which links are active.

- The FTS manager is not running. (F$MNOR)

You tried to stop the manager, but it is already inactive. To start the manager, use the FTOP -START_MNGR command.

- The server is not running. (F$SNOR)

You tried an FTOP -STOP_SRVR, -ABND_SRVR, or -ABRT_SRVR_LINK command when the server was not running.

- Server already running. (F$SRDL)

You tried to start a server that has already been started.

- Command must be invoked from system console. (F$SUSC)

You tried to start the manager from a terminal other than the system console.

- Text follows last argument. (F$TFLA)

This error occurs in FTGEN as well as in FTOP. The argument to a particular command contains extra text. For example, if you invoke the following command:

```
FTOP -STOP_MNGR FRED
```

the FTOP -STOP_MNGR command is acceptable, but FRED shouldn't be there as -STOP_MNGR does not require an argument.

- Unknown option. (F$UNOP)

You specified an unknown option in the command line. For example, the following is an invalid FTOP option:

```
FTOP UNICORN
```

This error occurs in FTR as well as in FTOP.

- Unexpected named semaphore value. (F$UNSV)

This error, although extremely unlikely, shows up after an FTOP -STOP_MNGR command. If it does happen, the FTS manager (YTSMAN) fails to close down. Your System Administrator should be informed, since the FTS manager can be shut down by logging out the YTSMAN phantom.

- You do not have operator privileges. (F$NOPP)

You attempted to invoke the FTOP command while not logged in as SYSTEM. You must log in as SYSTEM to use the FTOP command.

D

START_NET and STOP_NET Error Messages

INTRODUCTION

This appendix contains the error messages that may occur while you are using the START_NET and STOP_NET commands.

START_NET ERROR MESSAGES

The START_NET error messages are listed below.

- [nnnn Errors (Net_Init)]

This message prints out the total number of errors encountered while START_NET was processing. If nnnn equals 0, then the network will be started.

- Already Exists. The Network Was Already Started.
  You Must Use Stop_net Before Re-Starting The Network.

The network is already in operation. You must stop (STOP_NET) the network with the STOP_NET command before restarting it on the local node.

- A Nodename Must Be Specified With The -Node Option.

The START_NET command was given without specifying the name of the local node.

- A Nodename cannot be more than 6 characters long.

A node name longer than 6 characters was specified.

- Error: No path to node <name> in <nn> steps. Either recompile the INDIRECT_CONNECT subroutine with a larger MAX_STEPS parameter, or change the data file. Connection ignored (Net_Init).

This message states that there is no physical path between this node and the intended destination node. This indicates either that the node can't be seen at all, that there are more than 5 steps between end nodes, or that a longer path is needed.

- Error: PDN's not available with this version.

The PRINET product does not support PDNs.

- Error: PNC line spec has an invalid name <net> -> <Node>/<Line>

Ring configuration was not completely specified.

- Error: SMLC line spec has an invalid name "name"

SMLC line number was not configured, or was configured as unknown.

- Insufficient Access Rights.

Only System User May Invoke This Command.

START_NET was given from a terminal other than the supervisor terminal.

- Insufficient Access Rights. Can't Attach to PRIMENET*

The ACLs for PRIMENET* are incorrectly set.

- Insufficient Access Rights. PRIMENET*>NETWORK_SERVER.COMI

The ACLs for the server file are incorrectly set.

- Max addrs exceeded

The maximum number of addresses in the address table was exceeded.

- Max HCB's exceeded

The maximum number of HCBs was exceeded.

- Max names exceeded

The maximum number of names in the naming table was exceeded.

- Missing Argument to Command. -NODE nodename

The START_NET command was invoked without supplying the -NODE option.

- No room in alloc database

The network configuration is too big to fit into available memory. Try making the data base smaller and contact your support staff. You may get several of these messages.

- Not Found. Can't Attach to PRIMENET*

START_NET could not find the PRIMENET* directory.

- No Phantoms Available. Can't Start Network Server.

No phantoms were available when the START_NET command was given. Ensure that sufficient phantoms are configured before re-invoking START_NET.

- No Phantoms Available. Can't Start R-T Server

No phantoms were available when the START_NET command was given. The route-through server could not be started. Refer to the chapter on START_NET for a description of how to manually start R-T Server.

- Not Found. PRIMENET*>RT.COMI

START_NET could not find the file RT.COMI.

- Not Found. PRIMENET*>NETWORK_SERVER.COMI

START_NET could not find the file NETWORK_SERVER.COMI.

- Operation Illegal on Directory. PRIMENET*>NETWORK_SERVER.COMI

NETWORK_SERVER.COMI was created as a directory, instead of a file.

- Operation Illegal on Directory. PRIMENET*>RT.COMI

RT.COMI was created as a directory instead of a file.

- Too many paths specified

The maximum number of paths in the path table exceeded RING 0 limits.

- Warning: HDX not supported at this rev.

Half duplex is not supported at PRIMOS Revision 19.3.

- Warning: Node <name> does not have a specified <type> password. It will become a default value.

The specified type is NPX. Since passwords were not specified in the configuration file, default values are supplied.

STOP_NET ERROR MESSAGES

STOP_NET error messages are listed below:

- The Network Is Not Started Up

The STOP_NET command was given on a node that was not running on the network.

- No Right To Shutdown The Network

The STOP_NET command was given from a terminal other than the supervisor terminal.

INDEX

Index

A

Accepting connection requests, 14-14

Access rights for PRIMENET*, 18-3

Accessing remote files, 2-1

ADD_REMOTE_ID command, 2-4

Adding remote disks, 18-3

Architecture of PRIMENET, 10-1

Archiving FTS log files, 22-10

ARID command, 2-4

Assigning ports, 11-2, 14-4

B

Break in a ring,
  Locating, 23-28

C

Call Acceptance (IPCF), 14-14

Call Requesting (IPCF), 14-6

Canceling file transfer requests (FTS), 5-9

CCITT, 1-1

Clearing calls (IPCF), 14-20

Clearing codes, 11-6

Clearing virtual circuits, 11-6

Command mode of NETLINK, 7-3

Command summary of NETLINK, 9-1

Commands,
  ADD_REMOTE_ID, 2-4
  FTOP, 22-2
  FTR, 5-1
  LIST_REMOTE_ID, 2-5
  LOGIN, 3-1
  PRINT_NETLOG, 21-2
  REMOVE_REMOTE_ID, 2-5
  START_NET, 19-2
  STATUS NETWORK, 20-3

Commands (continued)
STATUS USERS, 20-2
STOP_NET, 19-4

Connection request,
Accepting, 14-14

D

Data network identification code, 9-8

Data transmission mode of NETLINK, 7-4

Destination site (FTS), 4-4, 6-6

DNIC, 9-8

E

Error messages,
FTGEN, C-2
FTOP, C-12
FTR, C-4
FTS, C-1
NETLINK, 9-17 to 9-21
PRINT_NETLOG, 21-14
Remote login, 3-2 to 3-4
START_NET, D-1
STOP_NET, D-4

Establishing remote ids, 2-4

Examining your remote ids, 2-5

F

Fast select calls, 15-9

File transfer requests (FTR),
Canceling, 5-9
Fetching, 5-3
Holding, 6-8
Notifying, 5-8
Printing, 5-3
Sending, 5-2
Status, 5-4

File transfer servers,
Listing, 22-8
Monitoring, 22-7
Starting, 22-8
Stopping, 22-8

File Transfer Service (FTS), 4-1

FIND_RING_BREAK,
Error messages, 23-30
How it works, 23-24
Input file, 23-26
Invoking, 23-26

Finding information on an incoming call (IPCF), 14-11

FT$SUB,
Defining keys, 17-3
Error codes, 17-3, 17-27
Error data structure, 17-31 to 17-33
Error recovery, 17-8, 17-13, 17-17, 17-21
Invoking, 17-4
Loading the library, 17-3
Program setup, 17-2
Programming example, 17-34 to 17-37

FT$SUB subroutine, 17-1

FTGEN, 4-1

FTOP command, 22-2 to 22-6
-ABND_SRVR option, 22-3
-ABRT_SRVR_LINK option, 22-3
-HELP option, 22-4
-LIST_SRVR_STS option, 22-4, 22-5
Listing file transfer servers, 22-8
Monitoring file transfer servers, 22-7
Monitoring YTSMAN, 22-7
-START_MNGR option, 22-5
-START_SRVR option, 22-5, 22-6
Starting file transfer servers, 22-7
-STOP_MNGR option, 22-6
-STOP_SRVR option, 22-6, 22-7
Stopping file transfer servers, 22-8

FTR,
Access rights, 4-6, 4-7
Canceling requests, 5-9
Destination pathname, 4-4
Destination site, 4-4
Destination user, 4-4
Failed transfers, 5-10
Fetching a file, 5-3
File transfer queues, 4-2
File transfer servers, 4-2
File types, 4-5
Full descriptions of management options, 6-14
Full descriptions of submittal options, 6-4
Help, 5-2
Introduction, 4-4
Logging Requests, 5-7
Management options, 6-14
Options for managing requests, 6-13 to 6-21
Options for submitting requests, 6-2 to 6-13
Printing a file, 5-3
Request name, 5-2
Request number, 5-2
Requesting transfer notification, 5-8
Sending a file, 5-2
Sitename, 5-2
Source pathname, 4-4
Source site, 4-4
Summary of management options, 6-14
Summary of submittal options, 6-2 to 6-4

FTR management options,
-ABORT, 6-15, 6-16
-CANCEL, 6-16
-DISPLAY, 5-5, 6-16, 6-17
-HELP, 6-17
-HOLD, 6-17, 6-18
-MODIFY, 6-18, 6-19
-RELEASE, 6-19
-STATUS, 6-20
-STATUS_ALL, 6-20, 6-21

FTR submittal options,
-COPY, 6-4, 6-5
-DELETE, 6-5
-DEVICE LP, 6-5
-DSTN_FILE_TYPE, 6-6
-DSTN_NTFY, 6-6

FTR submittal options (continued)
-DSTN_SITE, 6-6, 6-7
-DSTN_USER, 6-7
-HOLD, 6-8
-LOG, 6-8
-MESSAGE_LEVEL, 6-9
-NAME, 6-10
-NO_COPY, 6-10
-NO_DELETE, 6-10
-NO_DSTN_NTFY, 6-10
-NO_QUERY, 6-11
-NO_SRC_NTFY, 6-11
-QUERY, 6-11
-QUEUE, 6-11, 6-12
-SRC_FILE_TYPE, 6-12
-SRC_NTFY, 6-12
-SRC_SITE, 6-13
-SRC_USER, 6-13

FTS,
Error messages, C-1 to C-13
FTGEN error messages, C-2 to C-4
FTOP command, 22-2 to 22-7
FTOP error messages, C-12, C-13
FTR error messages, C-4 to C-12
Introduction, 4-1
Log files, 22-10
Monitoring, 22-10
Programming with, 17-1 to 17-37
Stopping, 22-11

FTSQ* directory, 22-10

G

General network cleanup (IPCF), 14-23

I

ICS1, 10-8

Interpreting ring statistics, 23-9

Intra-node calls (IPCF), 11-4

IPCF,
Checking return codes, 16-6
Descriptions, 14-4 to 14-34
Effect of START_NET and STOP_NET on, 16-8
Fast select calls, 15-9
Fast select example, 15-9 to 15-31
File-transmission system example, 15-3, 15-6 to 15-10
Front-end principles, 16-2
General layout, 15-2
Naming conventions, 14-2
Network event waiting, 16-5
Performance aspects, 16-3
Program closedown, 16-7
Programming examples, 15-1 to 15-31
Programming strategy, 16-1
Server principles, 16-2
Summary, 14-3
Timing aspects, 15-10
Virtual circuit clearing, 16-7
Window and packet sizes, 16-4

IPCF subroutines (See Subroutines)

L

LIST_REMOTE_ID command, 2-5

Locating a break in a ring, 23-28

Logging into remote systems, 3-1

LOGIN command, 3-1

M

Managing and monitoring user requests (FTS), 22-9

MDLC, 10-8

MONITOR_RING program,
Basic screen, 23-9
Invoking, 23-7

MONITOR_RING program (continued)
Report file format, 23-23
Selecting displays, 23-9

Monitoring,
FTS, 22-1
FTS log files, 22-10
FTSQ* directory, 22-10
NETMAN, 20-1
Network events, 21-1
Network servers, 20-1
RINGNET, 23-1
Route-through server, 20-2
YTSMAN, 22-7

N

NETLINK,
Address formats, 9-2
Addressing another system, 8-2
Basic commands, 9-2
Changing the prompt, 8-6
Command mode, 7-3
Command reference, 9-5 to 9-17
Command summary, 9-1 to 9-5
Connect packet options, 8-4
Data Transmission mode, 7-4
Debugging, 8-15
Direct remote login, 8-4
Displaying X.3 parameters (PAR command), 8-13
Error messages, 9-17 to 9-21
File transfers, 7-4, 8-7
International addressing, 8-3
Introduction to, 7-1
Invoking, 8-2
Literal addressing, 8-4
Local-to-remote file transfers, 8-8
Logging into a remote system, 8-4
Making a connection, 8-2
Multiple NETLINK connections, 8-5
PDN addressing, 8-3
PDNs and, 12-1
PRIMENET-configured name addressing, 8-3
Profile commands, 9-3
Remote-to-local file transfers, 8-9

NETLINK (continued)
Running it from a command input file, 8-9

NETLINK commands,
BPS, 9-5
C, 9-6
CALL, 9-7
CLEAR, 9-7
CLOSE, 9-7
CONTINUE, 9-7
D, 9-7
DATA, 9-8
DEBUG, 9-8
DNIC, 9-8
ESCAPE, 9-9
FCTY, 9-9
FILE, 8-7, 9-10
HELP, 9-11
LDATA, 9-11
LMDATA, 9-11
MDATA, 9-8, 9-11
MODE, 9-11
NC, 9-12
OUTFILE, 8-7, 9-12
PAR, 8-13, 9-12, 9-13
PAUSE, 9-13
POLL, 9-14
PORT, 9-14
PRID, 9-14
PROFILE, 8-11, 9-15
PROFILE DEFAULTS, 9-15
PROMPT, 8-6, 9-15
QUIT, 9-15
SET, 9-15
SPEED, 9-16
STATUS, 9-16
SW, 9-16
TO, 9-16, 9-17
TTP, 9-17

Network cleanup,
General (IPCF), 14-23

Network event file, 21-7

Network event messages, 21-8 to 21-15

Network programming, 13-1

Network servers,
Monitoring, 20-1

Network status interrogation (IPCF), 14-29

Network types, 10-4

O

Operator tasks,
Adding remote disks, 18-3
Controlling the network event file, 21-7
Interpreting ring statistics, 23-9
Invoking the FIND_RING_BREAK program, 23-26
Invoking the MONITOR_RING program, 23-7
Locating a break in a ring, 23-28
Managing user requests, 22-9
Monitoring FTS, 22-1
Monitoring NETMAN, 20-1
Monitoring network events, 21-1
Monitoring network servers, 20-1
Monitoring RINGNET, 23-1
Monitoring the FTSQ* directory, 22-10
Monitoring the Route-through server, 20-2
Monitoring user requests (FTR), 22-9
Rush requests, 22-9
Starting and monitoring file transfer servers, 22-7
Starting and monitoring YTSMAN (the FTS manager), 22-7
Starting PRIMENET, 19-1
Stopping FTS, 22-11
Stopping PRIMENET, 19-3

P

PDNs,
Connections, 10-9
Ease of access, 12-2
Multiple PDN support, 12-2
Route-through, 12-2

Point-to-point connections, 10-8

Ports, 11-1

PRIMENET,
  Adding remote disks, 18-3
  Architecture, 10-1
  Examples of networks, 10-5
  Invoking START_NET, 19-2
  Invoking STOP_NET, 19-4
  Level 1, 10-3
  Level 2, 10-3
  Level 3, 10-3
  Monitoring NETMAN, 20-1
  Monitoring network events, 21-1
  Monitoring network servers, 20-1
  Monitoring the Route-through server, 20-2
  Network configuration file, 18-3
  Network event file, 21-7
  Network programming, 13-1
  Network types, 10-4
  Operator features, 18-1
  PDNs, 10-9, 12-1
  Point-to-point connections, 10-8
  PRIMENET* directory, 18-2
  Programmer features, 13-1
  Remote file access, 2-1
  Remote login, 3-1
  RINGNET, 10-6
  Route-through connections, 10-9
  Sample ring, 23-25
  Starting and stopping, 19-1

PRINT_NETLOG command, 21-2 to 21-7
  Error messages, 21-14, 21-15

Printing files remotely (FTR), 5-3

Programming,
  FT$SUB example, 17-34 to 17-37
  IPCF examples, 15-1
  IPCF strategy, 16-1
  X.25 guidelines, A-1

R

Receiving Data (IPCF), 14-18

Releasing a port (IPCF), 14-23

Remote file access, 2-1

Remote ids,
  Establishing, 2-4
  Examining, 2-5

Remote login, 3-1

REMOVE_REMOTE_ID command, 2-5

Ring diagnostic programs, 23-1

RINGNET, 10-6, 23-1
  Configuration, 10-7
  Hardware, 10-6
  How it works, 23-5
  Invoking the FIND_RING_BREAK program, 23-26
  Invoking the MONITOR_RING program, 23-7
  Locating a break in a ring, 23-28
  Terminology, 23-2 to 23-5

Route-through connections, 10-9

Route-through server,
  Monitoring, 20-2

Rushing file transfer requests, 22-9

S

Sample ring, 23-25

Sending files with FTR, 5-2

Server processes (FTS), 4-2

Site (FTS),
  Defined, 4-4
  Destination, 4-4
  Local, 4-4
  Remote, 4-4
  Source, 4-4

START_NET command,
  Effect on IPCF programs, 16-8
  Error messages, D-1 to D-4
  Invoking, 19-2
  NET ON versus, 19-2

Starting a network, 19-1

Starting and monitoring file
    transfer servers, 22-7

Starting and monitoring YTSMAN,
    22-7

STATUS NETWORK command, 20-3

Status of FTR requests, 5-4

STATUS USERS command, 20-2

STOP_NET command, 19-3
  Effect on IPCF programs, 16-8
  Error messages, D-4
  Invoking, 19-4

Stopping a network, 19-1

Stopping FTS, 22-11

Subroutines,
  FT$SUB, 17-1 to 17-37
  IPCF, 14-1 to 14-34
  X$ACPT, 14-14
  X$ASGN, 14-4
  X$CLR, 14-20
  X$CLRA, 14-23
  X$CONN, 14-6
  X$FACP, 14-14
  X$FCLR, 14-20
  X$FCON, 14-6
  X$FGCN, 14-11
  X$GCON, 14-11
  X$GVVC, 14-25
  X$RCV, 14-18
  X$STAT, 14-29
  X$TRAN, 14-16
  X$UASN, 14-23
  X$WAIT, 14-24
  XLACPT, 14-14
  XLCONN, 14-6
  XLGCON, 14-11
  XLGVVC, 14-25

Summary of IPCF subroutines,
    14-3

System operator tasks, 18-1

T

Timed wait for call completion,
    14-24

Transferring files,
  With FTR, 4-2
  With NETLINK, 8-8

Transmitting data (IPCF), 14-16

U

Unassigning a port, 14-23

Using Remote ids, 2-2

V

Virtual circuits, 11-1
  Clearing, 16-7
  Clearing codes, 11-6 to 11-8
  Establishing, 15-1
  Passing off to other processes,
    11-5
  Status array, 11-5
  Throughput, 16-4
  Timing aspects, 15-10

W

Wait for completed network
    activity, 14-24

X

X.25,
  Call user data field
    terminology, A-1
  Data flow checkpoints, A-3
  Facility table, A-2
  Programming guidelines, A-1
  Window and packet size, A-2

Y

YTSMAN,
  Monitoring, 22-7

SURVEY

READER RESPONSE FORM

DOC3710-193 PRIMENET Guide

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Would you like to be on a mailing list for Prime's current documentation catalog and ordering information? ___yes ___no

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

PRIME

Attention: Technical Publications
Bldg 10B
Prime Park, Natick, Ma. 01760

READER RESPONSE FORM

DOC3710-193 PRIMENET Guide

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Would you like to be on a mailing list for Prime's current documentation catalog and ordering information? ___yes ___no

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

PRIME

Attention: Technical Publications
Bldg 10B
Prime Park, Natick, Ma. 01760

READER RESPONSE FORM

DOC3710-193 PRIMENET Guide

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Would you like to be on a mailing list for Prime's current documentation catalog and ordering information? ___yes ___no

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

PRIME

Attention: Technical Publications
Bldg 10B
Prime Park, Natick, Ma. 01760