60499500

CONTROL DATA CORPORATION

NETWORK PRODUCTS

NETWORK ACCESS METHOD VERSION 1/COMMUNICATIONS CONTROL PROGRAM VERSION 3 REFERENCE MANUAL

CDC® OPERATING SYSTEM:
NOS 2

REVISION RECORD

Revision	Description
A (12/01/76)	Original Release. PSR Level 439.
B (04/01/77)	Revised to PSR Level 446 for technical corrections.
C (07/01/77)	Revised to PSR Level 452 for technical corrections.
D (04/28/78)	Completely revised for NAM version 1.1 release at PSR level 472 to include support of remote and foreign NPUs, asynchronous and HASP TIPs, virtual terminals, IAF, and TVF.
E (08/15/78)	Revised at PSR level 477 for technical corrections.
F (12/18/78)	Revised at PSR level 485 for technical corrections.
G (01/15/79)	Revised at PSR level 485 for additional technical corrections.
H (08/10/79)	Revised to reflect release of NAM version 1.2. Included are descriptions of the binary debug log file and postprocessor, special editing support, and QTRM.
J (12/11/79)	Revised to reflect addition of connection duplexing, upline block truncation, block header break markers, QTRM connection switching, and various technical corrections.
K (04/18/80)	Revised at PSR level 517 to reflect the addition of 714 printer support, and various technical corrections.
L (10/31/80)	Revised at PSR level 528 to reflect the addition of QTRM support of application-to-application connections, the user-interrupt capability, and various technical corrections.
M (05/29/81)	Revised for NAM Version 1.3 release at PSR level 541 to include 2780/3780 terminal support, changes to supervisory messages, PRU interface, and various technical corrections.
N (02/26/82)	Revised at PSR level 559 to reflect release of NAM Version 1.4, which supports NOS Version 2.0 and includes the disable flag parameter on the LST/HDX/R supervisory message and miscellaneous technical corrections.
P (01/14/83)	Revised at PSR level 580 to reflect release of NAM Version 1.5 and CCP Version 3.5, which run only under the NOS Version 2 operating system. This manual, which was previously known as the NAM Reference Manual, is no longer applicable to products operating under NOS 1. It has been reorganized to document information needed by a general networks user, who must consider NAM as well as CCP when writing a network application. This is a complete reprint.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Title Page	-
ii	P
iii/iv	P
v	P
vi	P
vii/viii	P
ix thru xii	P
xiii	P
1-1 thru 1-6	P
2-1 thru 2-35	P
3-1 thru 3-43	P
4-1 thru 4-18	P
5-1 thru 5-19	P
6-1 thru 6-15	P
7-1 thru 7-32	P
8-1	P
A-1 thru A-48	P
B-1 thru B-10	P
C-1 thru C-17	P
D-1	P
D-2	P
E-1 thru E-30	P
F-1 thru F-16	P
G-1	P
G-2	P
H-1 thru H-12	P
I-1 thru I-22	P
Index-1 thru -4	P
Comment Sheet	P
Mailer	-
Back Cover	-

PREFACE

This manual, formerly known as the Network Access Method reference manual, has been renamed the NAM Version 1/CCP Version 3 reference manual. Its new organization supplies reference information to both Network Access Method (NAM) and Communications Control Program (CCP) users, typically either systems programmers or analysts who are writing a network application or who would like to learn more about how the various portions of the network fit together.

This book describes how applications interface to the computer network and how the terminal user gains access to these applications. Plus, this book familiarizes the reader with the Network Processing Unit (NPU) and the Communications Control Program (CCP). Knowledge of the NPU and CCP, however, is not necessary to write an application program.

NAM and CCP operate under control of the NOS 2 operating system for the CONTROL DATA® CYBER 170 Computer Systems; CDC® CYBER 70 Computer System models 71, 72, 73, and 74; and 6000 Computer Systems.

NAM is the subset of the host computer software that provides communication between an application program in the host computer and other application programs or devices accessing the network's resources.

The Communications Control Program is software that resides in a 255x Series Network Processing Unit that allows a device to access the host computer over communications lines.

WHO SHOULD READ THIS MANUAL

This manual is directed at the general networks user (a systems programmer or analyst), who is familiar with subsystem applications programming, compiler and assembler programming conventions, terminal communication protocols, other network software products, and the programming requirements of supported devices.

HOW THIS MANUAL IS ORGANIZED

In addition to describing NAM, this book now contains the majority of the information previously found in the CCP reference manual. Information has been consolidated into this book to reduce redundancy and to provide a central location for information needed by the general networks user.

Section 1 introduces the NAM and CCP software. Section 2 describes the protocols governing information exchanged for communication between NAM and each application program, and between application programs and their connections. Section 3 describes the synchronous and asynchronous supervisory messages used by application programs. Section 4 describes the language and internal interfaces required by an application program. Section 5 discusses the application interface program statements used by NAM to access the network and to send and receive messages. Section 6 discusses the structure and execution of an application program job as a batch or system origin type file. Section 7 gives more detailed information on CCP than was presented in the first section of the book. Section 8 describes network failure and techniques of recovery.

Other parts of the Communications Control Program reference manual have been moved to other network product and operating system publications. Use table 0-1 to help find the location of this information.

RELATED PUBLICATIONS

Related material is contained in the publications listed below. Other manuals may be needed, such as the hardware, firmware, or emulator software reference manual for the devices serviced by a given program. Also, communication standards and device operating literature can be useful.

The NOS 2 Manual Abstracts is a pocket-sized manual that contains a brief description of the contents and intended audience of every manual available for NOS 2 and its product set. The abstracts manual can help a particular reader determine which manuals are of greatest importance.

Another manual, the Software Publications Release History, gives the titles and revision levels of software manuals available for the Programming System Report (PSR) level of NOS 2 installed at your site.

TABLE 0-1. NEW LOCATION OF CCP REFERENCE MANUAL INFORMATION

Information	Manual That Contains Information				
	NOS Version 2 NAM/CCP Reference Manual	NOS Version 2 System Maintenance Reference Manual	Communications Control Program Version 3 Diagnostic Handbook	NOS Version 2 Operator/ Analyst Handbook	Communications Control Program Internal Maintenance Specification†
CCP overview, concepts, and functions	X				
NPU initialization, failure, and recovery	X				
Character sets	X				
CCP glossary	X				
Mnemonics	X				
Statistics		X			
Halt Codes		X	X		
Diagnostics			X		
Customer Engineering error messages			X		
Dump information			X		
NPU operating instructions				X	
Memory map					X
Naming conventions					X
NPU dumping, loading, and initializing details					X

†Available from Software Manufacturing Distribution (SMD), 4201 Lexington Ave. North, Arden Hills, Minnesota 55112

The following manuals are of primary interest:

Publication	Publication Number
COMPASS Version 3 Reference Manual	60492600
Network Products Network Access Method Version 1 Network Definition Language Reference Manual	60480000
Network Products Network Access Method Version 1 Terminal Interface Guide	60480600
Network Products Remote Batch Facility Version 1 Reference Manual	60499600
NOS Version 2 Reference Set, Volume 1 Introduction to Interactive Usage	60459660

NOS Version 2 Reference Set, Volume 3 System Commands	60459680
NOS Version 2 Reference Set, Volume 4 Program Interface	60459690

The following manuals are of secondary interest:

Publication	Publication Number
CCP Version 3 Diagnostic Handbook	60471500
COBOL Version 5 Reference Manual	60497100
FORTRAN Extended Version 4 Reference Manual	60497800
FORTRAN Version 5 Reference Manual	60481300
Message Control System Version 1 Reference Manual	60480300
Network Processor Unit Hardware Reference Manual	60472800
NOS Version 2 Diagnostic Index	60459390
NOS Version 2 Installation Handbook	60459320
NOS Version 2 Manual Abstracts	60485500
NOS Version 2 Operator/Analyst Handbook	60459310
NOS Version 2 System Maintenance Reference Manual	60459300
Software Publications Release History	60481000
TAF Version 1 Reference Manual	60459500
255x Host Communications Processor/ Network Processor Unit Reference Manual CCP Version 1.0	60470000
255X Network Processor Unit Hardware Reference Manual	60472800
2560 Series Synchronous Communications Line Adapter Hardware Maintenance Manual	74700700
2561 Series Asynchronous Communications Line Adapter Hardware Maintenance Manual	74700900
2563 Series SDLC Line Adapter Hardware Maintenance Manual	74873290

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

NOTATIONS

Throughout this manual, the following conventions are used in the presentation of statement formats, operator type-ins, and diagnostic messages:

UPPERCASE	Uppercase letters indicate acronyms, words, or mnemonics either required by the network software as input, or produced as output.
lowercase	Lowercase letters identify variables for which values are supplied by the NAM or terminal user, or by the network software as output.
...	Ellipsis indicates that omitted entities repeat the form and function of the entity last given.
[]	Square brackets enclose entities that are optional; if omission of any entity causes the use of a default entity, the default is underlined.
{}	Braces enclose entities from which one must be chosen.
input parameter	This term identifies an AIP call statement parameter for which values are supplied to AIP by the programmer.
return parameter	This term identifies an AIP call statement parameter for which variables are supplied to AIP by the programmer and in which values are placed by AIP.
<ct>	The ct symbol represents the network control character defined for the terminal. This character must be the first character of the command entered.
LF	The LF symbol represents a one-line vertical repositioning of the cursor or output mechanism. LF also designates a character or character code associated with such a line feed operation.
(c)	A circle around a character represents a character key that is pressed in conjunction with a control key (CTL, CNTRL, CONTRL, CONTROL, or equivalent).
CR	The boxed CR symbol represents the terminal key that causes message transmission; usually, this is the same key that causes a carriage return operation. Transmission keys are described in more detail in appendix F.

Unless otherwise specified, all references to numbers are to decimal values and all references to bytes are to 8-bit bytes and all references to characters are to 7-bit ASCII-coded characters. Fields defined as unused should not be assumed to contain zeros.

NETWORK PRODUCTS: AN OVERVIEW

This section introduces the Control Data Corporation CYBER 170 network products, their relationships to each other, and their significance to the data communications user. Network products is a group of programs and hardware that provides communications services to geographically dispersed users.

As shown in figure 1-1, a network consists of a computer network and a communications network.

COMPUTER NETWORK

The computer network includes host computers, application programs, terminals, and the host software associated with network communications. Each application program gives the terminal user or application a specific data processing capability.

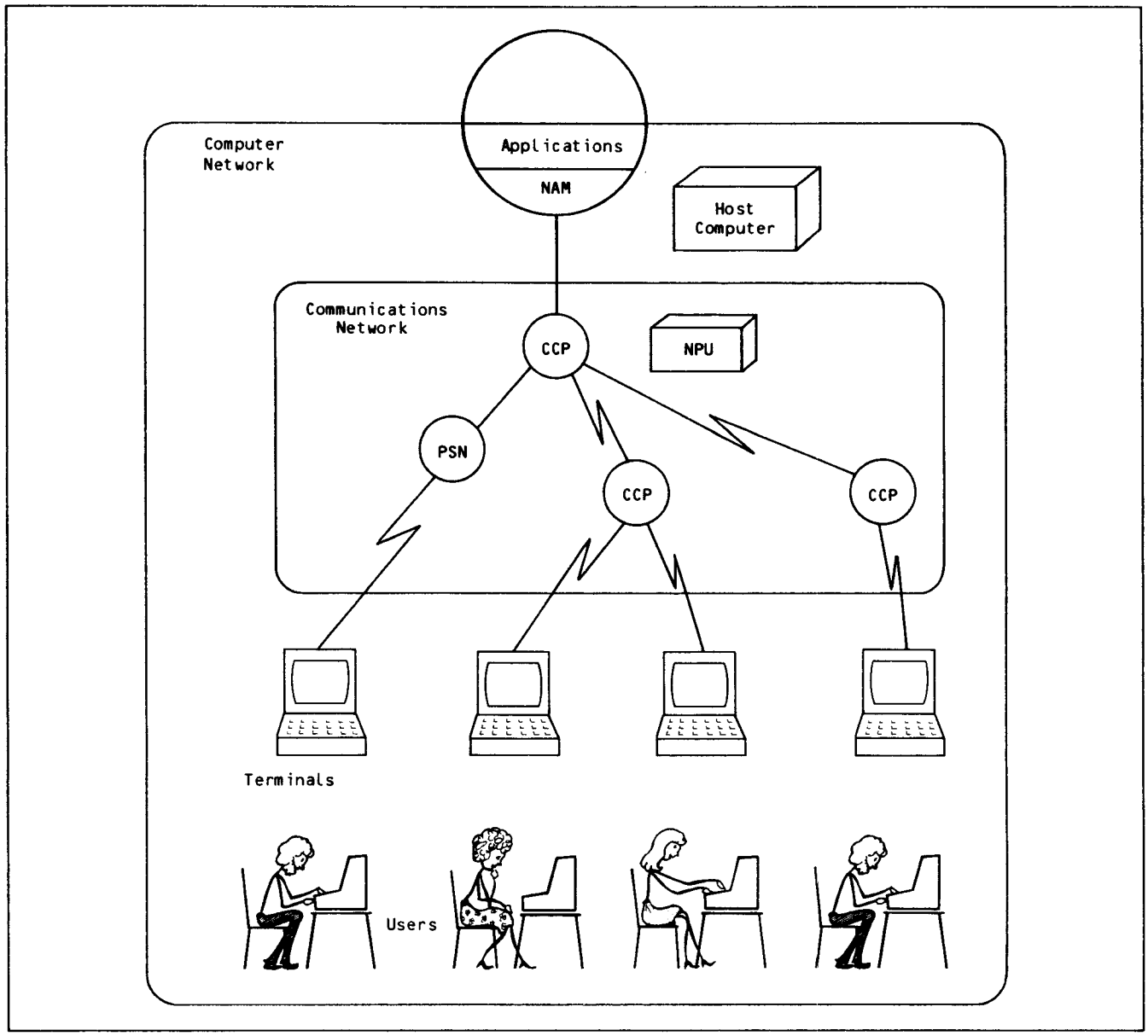


Figure 1-1. Overview of Network Products

COMMUNICATIONS NETWORK

The communications network includes network processing units (NPUs) and the connecting communication lines and packet switching networks (PSNs) needed to transport blocks of data between host computers and terminals.

The size and complexity of a communications network varies from a simple network with one local (front-end) NPU or a network with one local NPU and one or more remote NPUs to a more complex network with multiple local NPUs and multiple remote NPUs. Attached to these NPUs are terminal devices, such as entry/display stations.

Because the communications network minimizes terminal type dependency and removes many of the terminal switching operations from the host, the host can process data more efficiently.

SOFTWARE COMPONENTS OF THE NETWORK

Figure 1-2 shows the interfaces between the elements of the network. The left part of the figure shows the network host products (NHP), which are the software elements located in the CDC CYBER 170 host computer. The middle section shows the Communications Control Program (CCP), which is the software element located in the Network Processing Unit. As shown in the right portion of figure 1-2, CCP communicates directly with the terminals while the Network Access Method (NAM) communicates with applications. Refer to figure 1-2 while reading the remainder of this overview section on network products.

NETWORK HOST PRODUCTS

Network host products includes the network access software and the application programs that provide the specific facilities requred by terminal users.

The network access software is collectively called the Network Access Method or NAM. NAM is used in several contexts throughout this manual and in the other network products documentation. NAM can refer to the interface between application programs and the communications network; to the programs that implement that interface, including the Applications Interface Program (AIP), the Network Interface Program (NIP), and the Peripheral Interface Program (PIP); or to the product NAM, which also includes the Network Supervisor (NS), the Communications Supervisor (CS), and the Network Validation Facility (NVF).

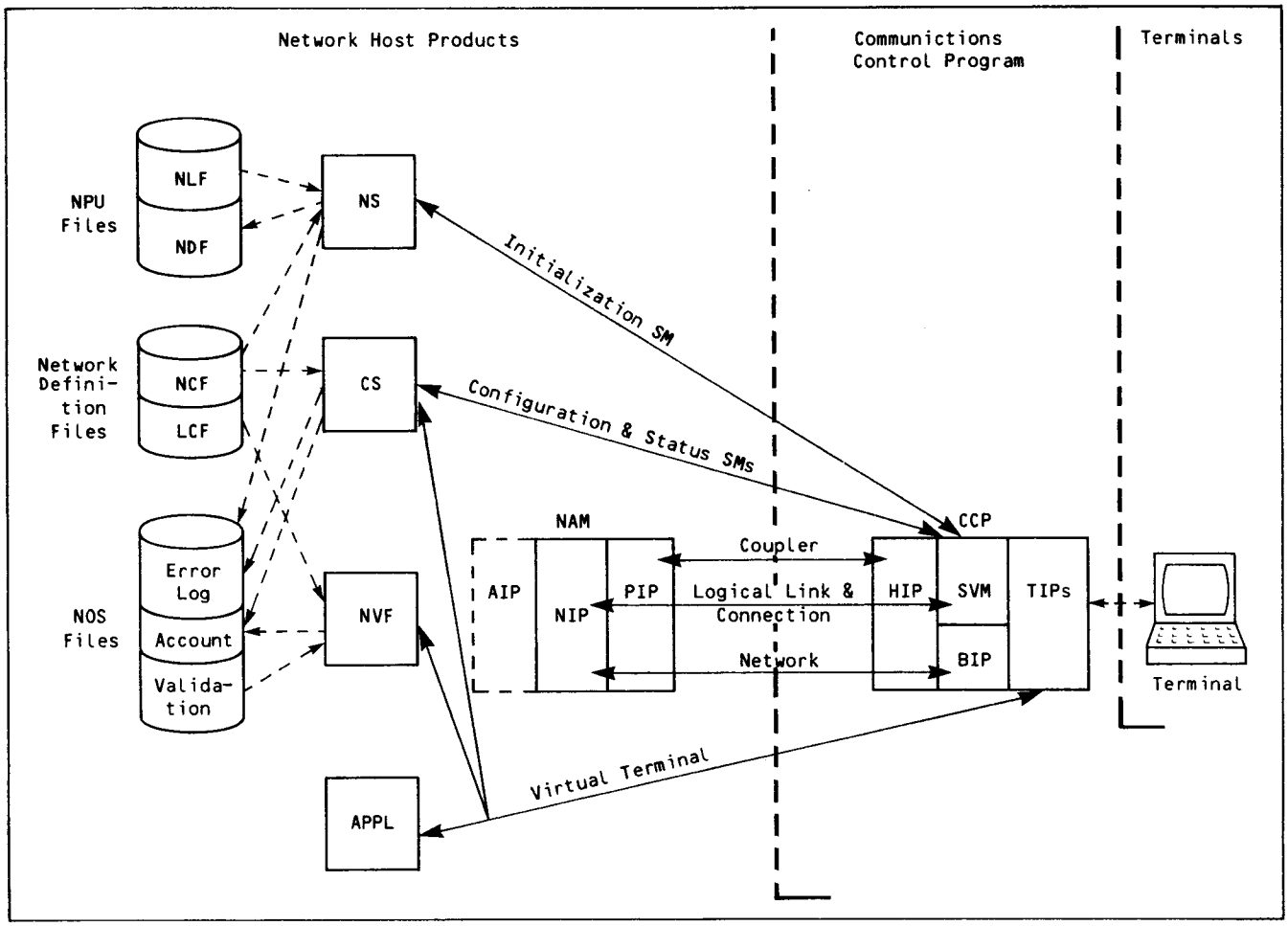


Figure 1-2. The Interfaces Between the Network Product Elements

In figure 1-2, NAM refers to the set of programs that implement the interface between the application programs and communications network.

Network host products sofware, shown in the left part of figure 1-2, includes:

Network Access Method

Network Definition Language Processor

Network Supervisor

Communications Supervisor

Network Validation Facility

Network utilities

Network Access Method application programs

NETWORK ACCESS METHOD

The Network Access Method is the central network host product. NAM interfaces between applications in the same host or between applications and the Communications Control Program.

NAM software consists of three interface programs that provide a common way for CDC CYBER 170 applications to access the communications network. NAM resolves resource contention among application programs and buffers data to regulate data flow.

Peripheral Interface Program

The Peripheral Interface Program (PIP) is a peripheral processor unit program that interfaces the central processor executed routines of NAM to the channel-connected local NPUs.

PIP moves blocks of data between the central memory buffers of NAM and the NPU and reads and writes disk files used by batch devices. PIP also can detect when a local NPU needs initializing. If the NPU does not have system autostart module hardware, PIP requests the network supervisor to load the bootstrap program into the NPU.

Network Interface Program

The Network Interface Program (NIP) executes as a system control point. NIP coordinates the use of the communications network by all application programs, buffers data between the application programs and the network, and manages the logical connections.

The buffering provided by NIP eliminates the need for application programs to have outstanding buffers. Although an application program is expecting input data, the application program can be swapped out, which reduces the use of host resources.

Each application program may have several connections; each connection is associated with a terminal device or with another application program. NIP translates between network addresses and the more convenient logical addresses that represent the connection to the application. NIP also establishes new connections as they are requested and terminates connections that are no longer needed or that have failed.

An application may request NAM to convert the data on a logical connection from the network format. Such conversions determine the format and encoding of characters seen by the application.

Application Interface Program

The Application Interface Program (AIP) is a set of subprograms that resides in the application program's field length and provides a procedural interface to the capablilities of NIP and the network.

Procedural statements are provided so that the application program can connect to and disconnect from the network. Procedural statements also control information exchange between the application program and NAM buffers. This information may be data, or it may be supervisory messages that coordinate the application's execution with events that have occurred in the network. NAM may pass a supervisory message to inform the application of a new connection that is requesting service, or that a failure has occurred. In the same way, the application program uses supervisory messages to communicate with NAM and the network elements.

Queued Terminal Record Manager

Queued Terminal Record Manager (QTRM) is a set of procedures that resides in the application program's field length and provides a high level procedural interface to the network. QTRM is discussed in appendix E of this book.

NETWORK DEFINITION LANGUAGE PROCESSOR

Before the network software can route data through the network and interface to operators for supervision, the definition of the network configuation must first be communicated to the software. The Network Definition Language (NDL) is used to describe this configuration. The Network Definition Language processor (NDLP), an off-line batch utility, translates this configuration and prepares a network configuration file (NCF) and a local configuration file (LCF).

The NCF contains network configuration information required by the network.

The LCF contains host information required by the Network Validation Facility, such as automatic log-in parameters and application information. The LCF allows the network validation facility to validate and connect to applications.

NDL is described in the Network Definition Language reference manual.

NETWORK SUPERVISOR

The Network Supervisor (NS), which executes as a NAM application, interfaces between the NPUs and CCP program files in the host. NS responds to requests to load NPUs with their software and saves NPU dumps on host files.

COMMUNICATION SUPERVISOR

The Communication Supervisor (CS) program executes as a NAM application. It interfaces to the network processing unit operator (NOP). CS allows a network operator at a terminal or host console to obtain and change the status of network elements, to communicate with users at terminals, and to run diagnostics. CS also responds to requests for network configuration data from an NPU.

NETWORK VALIDATION FACILITY

This program, which executes as a NAM application, validates the terminal user's access to the host and an application program's access to the computer network. The Network Validation Facility (NVF) also maintains and reports application status to the host operator (HOP). As figure 1-2 shows, the NOS validation file and the local configuration file (LCF) supply validation information.

NETWORK UTILITIES

Four utility programs either are included with or used by network host products:

The Network Dump Analyzer (NDA)

The Load File Generator (LFG)

The Debug Log File Processor (DLFP)

The Hardware Performance Analyzer (HPA)

Network Dump Analyzer

This host utility produces a formatted printout from NPU dump files created by the Network Supervisor. The site analyst can use these dumps to help analyze CCP software or NPU hardware failures. The network dump analyzer uses the network dump file (NDF), which is shown in figure 1-2, as input.

Load File Generator

This host utility reformats CCP program files produced by the CDC CYBER Cross System's link and edit programs into a single random access file used by the Network Supervisor to load NPUs. This file is the network load file (NLF), which is one of the NPU files shown in figure 1-2.

Debug Log File Processor

This host utility processes the debug log file generated by the Application Interface Program.

Hardware Performance Analyzer

A fourth utility program, the hardware performance analyzer (HPA), is part of the NOS operating system. This CYBER utility program produces reports from information on the account and error log dayfiles. Network products software makes statistical, error, and alarm message entries into these dayfiles.

NAM APPLICATION PROGRAMS

The host computer executes CDC-written or site-written service programs called application programs that are connected to the network through NAM. An application program can communicate with other application programs or terminals connected to the network.

The CDC-provided NAM application programs are:

Interactive Facility (IAF), which allows you to create files and to create or execute programs from a device without using card readers or line printers. IAF is described in Volumes 1 and 3 of the NOS 2 Reference Set.

Remote Batch Facility (RBF), which permits you to enter a job file from a remote card reader and to receive job output at a remote batch device. RBF is described in the Remote Batch Facility reference manual.

Transaction Facility (TAF), which permits you to implement on-line transaction processing under NOS by writing programs to use terminals. TAF is described in the TAF reference manual.

Terminal Verification Facility (TVF), which provides tests you can use to verify that an interactive console is sending and receiving data correctly. TVF is discussed in the Terminal Interface Guide.

Message Control System (MCS), which allows you to queue, route, and journal messages between COBOL programs and terminals. MCS is described in the Message Control System reference manual.

NETWORK PROCESSING UNIT AND COMMUNICATIONS CONTROL PROGRAM

This subsection discusses the following network products, which are part of the communications network and allow a terminal to access the host computer over communication lines:

The 255x Series Network Processing Unit (NPU), which connects a host to a terminal

The Communications Control Program (CCP), which is the software in the NPU

The CDC CYBER Cross System, which supports installing, maintaining, and modifying CCP

The middle portion of figure 1-2 shows the communications network.

NETWORK PROCESSING UNIT

An NPU handles front-end or remote data communications for the CDC CYBER 170 host. The Communications Control Program resides within the NPU.

To understand CCP, you must have a basic understanding of the hardware on which CCP runs. Refer to the hardware manuals listed in the preface for a description of the hardware components of the NPU.

COMMUNICATIONS CONTROL PROGRAM

The Communications Control Program, which is the software that executes in the 255x NPUs, consists of:

- Base system software
- Service Module
- Host Interface Program
- Terminal Interface Programs
- Link Interface Program
- Block Interface Program
- In-line and on-line diagnostics
- Console debugging aids
- Performance and statistics programs

Figure 1-3 shows how the major parts of CCP relate to each other.

Base System Software

The base system software executes programs, allocates buffers, handles interrupts, and supports timing and data structures. It includes:

- A system monitor, which controls the allocation of resources for the communications processor
- Timing services, which run those programs or functions that are executed either periodically or following a specific time lapse for the processor
- A multiplex subsystem, which interfaces with the multiplex hardware and performs character by character processing of tasks
- Interrupt handler, which controls the transition of the communications processor between different program interrupt levels
- Initialization, which prepares the network for on-line operation
- Structure services, which build and maintain internal tables used for routing data
- Buffer maintenance, which dynamically allocates memory in multiple buffer sizes for efficient memory use
- Worklist services, which provide logic for 255x interprogram communication through the use of worklists
- Standard subroutines, which provide support routines to handle arithmetic conversion, maintain page registers, and do miscellaneous tasks

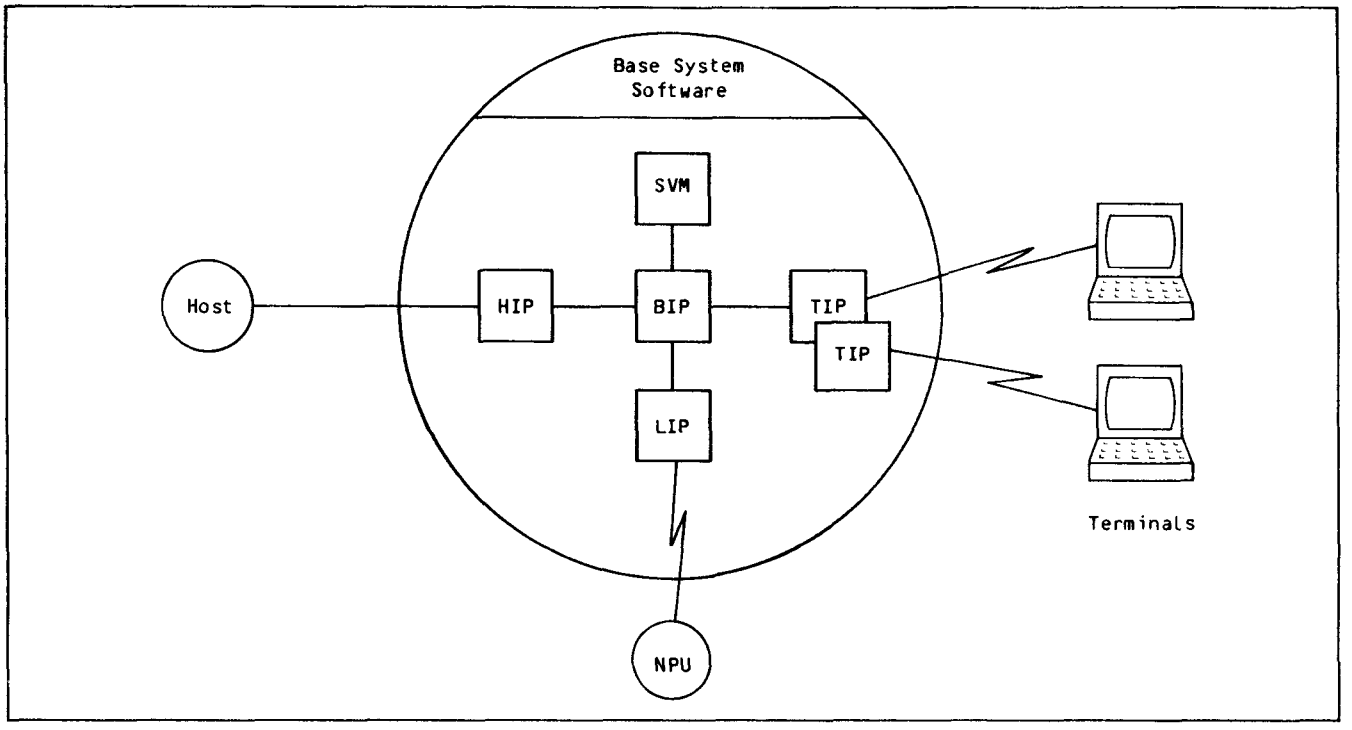


Figure 1-3. The Relationship Between the Parts of the Communications Control Program

Service Module

The Service Module (SVM) includes network control functions and interface programs that provide a common link to other elements of the communications network. These programs:

Process service messages

Control line and terminal configuration

Report and respond to regulation and supervision changes

Host Interface Program

The Host Interface Program (HIP) provides the software that links the host and a local NPU over a channel. The HIP drives the CDC CYBER Channel Coupler, transfers data, checks for errors, and monitors for host failure and recovery.

Terminal Interface Program

The Terminal Interface Program (TIP) is a modular program that provides protocol support and the control needed to interchange data between a terminal and other elements of CCP. CDC provides TIPs for asynchronous, X.25, mode 4, bisynchronous, and HASP terminals. Non-CDC TIPs may be written to support specific site requirements. Section 7 discusses TIP details.

Link Interface Program

The Link Interface Program (LIP) transfers information over a trunk between NPUs.

Block Interface Program

The Block Interface Program (BIP) routes blocks of data, processes service messages, and processes the network block protocol.

In-Line and On-Line Diagnostics

In-line and on-line diagnostics, which are produced for the NPU, enable a NOP to isolate communications line problems. Alarm, CE error, and statistics service messages are the types of in-line diagnostics. In-line diagnostics are generated automatically. On-line diagnostics must be requested from the NOP console.

Console Debugging Aids

Debug aids provide test utilities for debugging programs, taking memory snapshots, and dumping the NPU during program development or system failures.

Performance and Statistics Programs

These programs gather statistics on NPU and individual line performance, and periodically dispatch theses statistics to the Communications Supervisor.

CDC CYBER CROSS SYSTEM SOFTWARE

The CDC CYBER Cross System software allows you to install, modify, and maintain the CCP software. It is composed of these programs:

PASCAL, which is a high level compiler patterned after ALGOL-60. By using PASCAL, you can define tasks in statements that are processed by the compiler to yield a variable number of actual program instructions.

Formatter, which reformats PASCAL output into an object code format compatible with the communications processor macro assembler output

Macro Assembler, which assembles communications processor macro level source programs and produces relocatable binary output. The source programs are written with symbolic machine, pseudo, and macro instructions.

Micro Assembler, which provides the language needed to write a micro program. This assembler translates symbolic source program instructions into object machine instructions.

Link Editor, which accepts object program modules and generates a memory image, suitable for executing in the 255x NPU.

Autolink Utility, which simplifies program assignment and maximizes the amount of space assigned to handling buffers.

Expand Utility, which includes several hardware and software variables used to define a CCP load file for a given NPU configuration.

See the preface for manuals that contain more information on the CDC CYBER Cross System.

INFORMATION PROTOCOLS 2

This section describes the protocols governing information exchanged for communication between the Network Access Method (NAM) and each application program, and between application programs and their connections. The first portion of this section defines the terms and concepts needed to understand the description of information content in the remainder of this section.

You should remember that parts of the network software are written as application programs and also use these protocols. Some of the features and options discussed in this and subsequent sections, therefore, do not necessarily apply to site-written application programs; such information is indicated where it is described.

INFORMATION FLOW

Information flow in the network is defined from the viewpoint of the host computer. Information coming to the host is said to be traveling upline; information moving away from the host is said to be traveling downline.

STRUCTURE PROTOCOLS

The network software uses structure protocols of two types:

A logical protocol based on the concept of a message

A physical protocol based on various definitions of a block of data

The conditions that create a logical message and the conventions governing the subdivision of messages are influenced by the physical structure protocols the network uses. The events involved in actually creating a message are described later in this section under the headings Interactive Terminal Input Concepts and Interactive Terminal Output Concepts.

PHYSICAL PROTOCOLS AND NETWORK BLOCKS

Information exchanged with the network is either data of no significance to the network software or nondata control information of significance only to the network software. Exchanges of control information and data between application programs, the network software, and a terminal user occur in logical messages comprising one or more physical network blocks. A network block is a physical subdivision of a logical entity.

A network block is a grouping of information with known and controllable boundary conditions, such as length, completeness of the unit of communication, and so forth. Other network documentation refers to network blocks as network data blocks; this manual uses the term data block only when referring to network blocks that do not contain control information.

Information exchanges between network processing units and host computers or between application programs use this physical structure protocol. Such exchanges occur in single network blocks.

Information exchanges between network processing units use a different physical structure protocol. Such exchanges occur in sets of character and control bytes called frames. The relationship of a frame to a network block is not significant to an application programmer; frames are not discussed in this section.

Information exchanges between network processing units and terminal devices use a third physical structure protocol. Such exchanges occur in sets of character and control bytes called transmission blocks.

LOGICAL PROTOCOL AND PHYSICAL BLOCKS

Upline and downline information within the host and NPUs is always grouped into physical network blocks. Network data blocks are grouped into logical messages. Messages exchanged between an NPU and a device can also be grouped into physical transmission blocks of one or more logical messages. Figure 2-1 shows these concepts.

Network blocks are restructured into other types of blocks at points of entrance and exit from the network processing units. Figure 2-2 shows these points as circles.

Network Data Blocks

A network data block is a collection of character bytes, analogous to a clause in English. It is a partially independent unit of information and might need to be used with other blocks to form a message.

A network data block can contain all or part of a message. Whether a message must be divided into several network data blocks is determined by the size of a network data block.

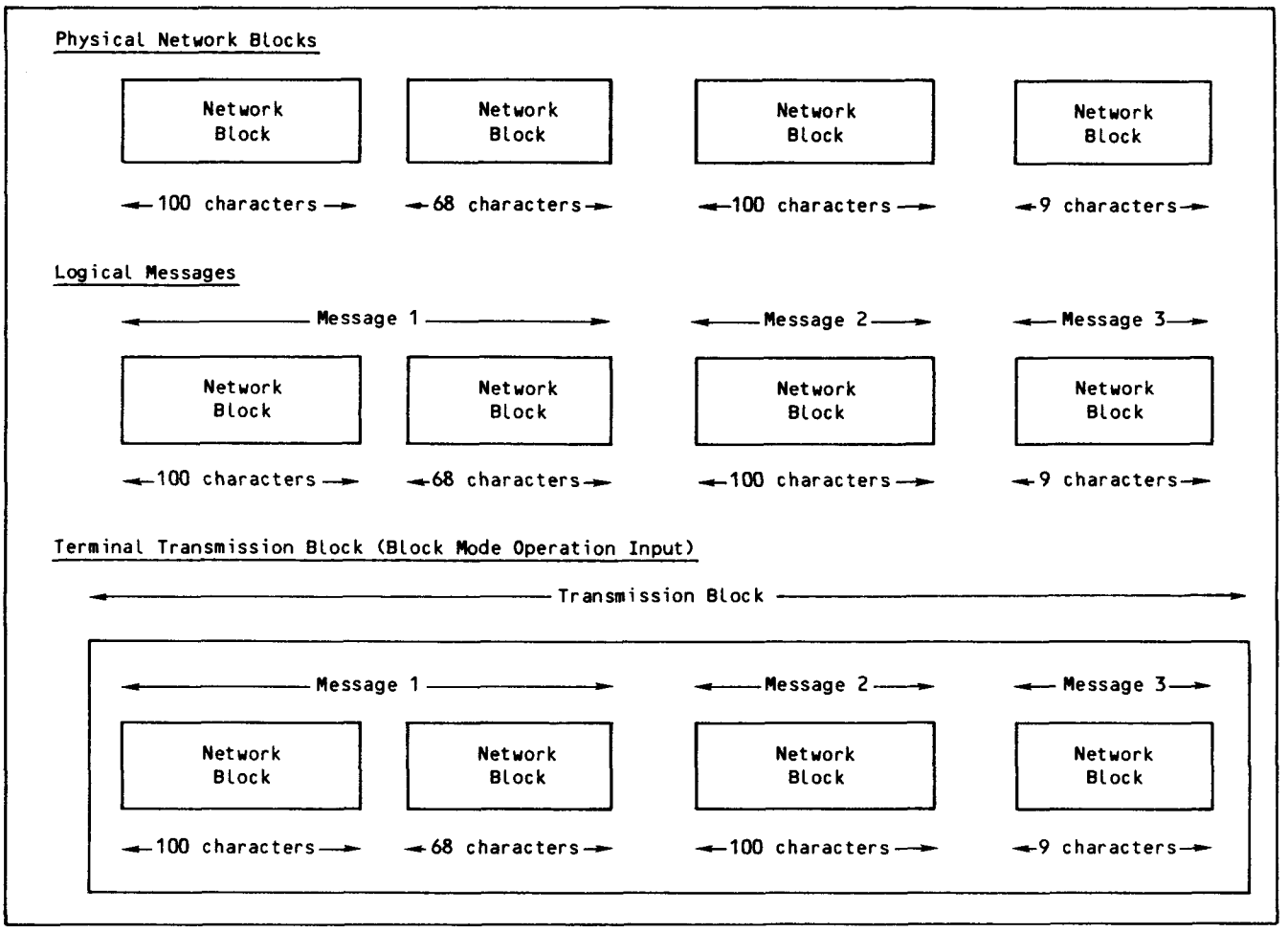


Figure 2-1. Physical and Logical Information Structures

Upline and Downline Block Sizes

CDC-defined interactive devices have network data block sizes that are multiples of 100 character bytes for upline data and of varying sizes for downline data. The last block of an upline message need not contain a multiple of 100 characters.

CDC-defined batch devices have network data block sizes that are multiples of 640 central memory words. Each such block is one mass storage physical record unit (PRU) of a file.

The network administrator establishes the appropriate size of upline and downline network data blocks for each terminal device when the network configuration file is created. Sizes are usually chosen to fit a single message into a single network data block, or to optimize use of available network storage, or to satisfy some other administrative criterion. The administrator also establishes the correct size for a terminal transmission block in the network configuration file.

The initial size of an upline network data block is established by the site administrator (using the UBZ parameter of an NDL statement) when he or she defines the device that produces the block. Once a size is established for a device, that size determines the maximum number of characters an application program can receive as a single network data block. When an upline message is too long to fit into a single network data block, the NPU divides it into as many network data blocks as necessary before delivery to the application program.

The initial size of a downline network data block is established by the site administrator (using the DBZ parameter of an NDL statement) when he or she defines the device that receives the block. The established size is a recommended maximum for the number of characters an application program should send in a single network block. The actual maximum size of a downline network block is chosen by the application program sending the block. NAM imposes an absolute maximum size, however; this absolute maximum is described later in this section under the heading Block Buffer Areas.

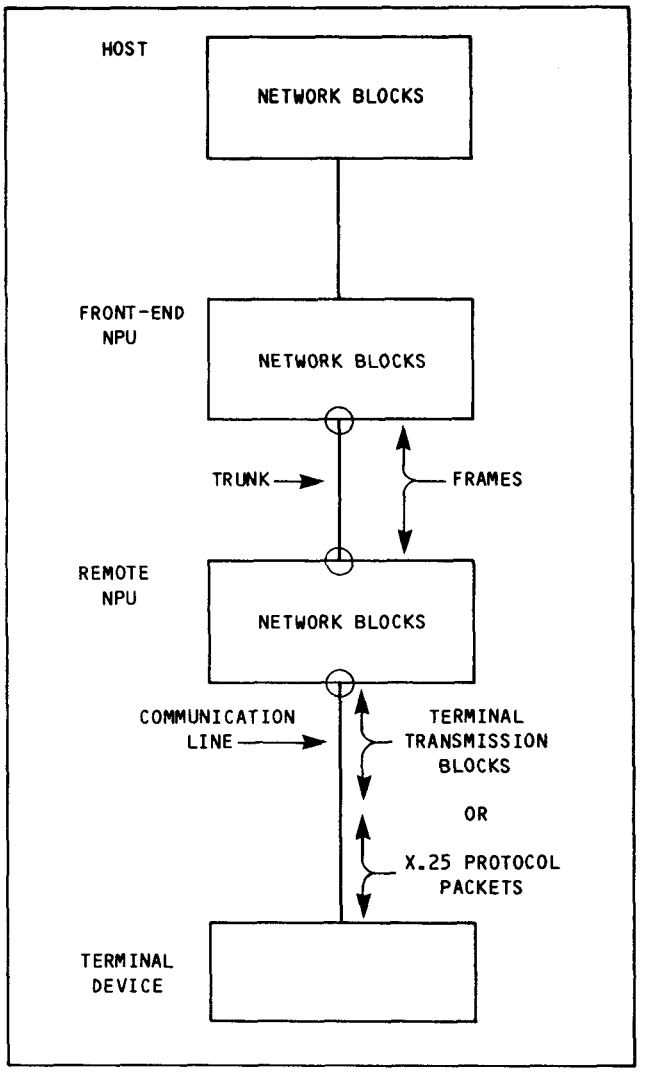


Figure 2-2. Block Reassembly Points

The maximum length used for each network data block to or from a device can be independent of the terminal's transmission block size. For example, a mode 4 console cannot accept a transmission block containing more than a specified number of characters. An application program could divide a multiple line display transmitted to the console of such a terminal into network blocks smaller than the buffer space of the specific terminal. However, the application program does not need to divide its network blocks. The network software reconstructs any of the program's network data blocks longer than the terminal's buffer space into several terminal transmission blocks of the correct size.

An application program is advised of the upline and downline network data block sizes defined for each device it services when logical connection to a device occurs. Your application program can change the established upline block size using control information called a field number/field value pair; this process is described in section 3. Your application program cannot change the established downline block size but can ignore it.

The upline block size is enforced by the network software, which subdivides terminal transmission blocks input from a device into network data blocks of that size or smaller. The upline block size defines the largest block that NAM will deliver to the application program.

The downline block sizes defined are advisory values. That is, an application program can accept the size specified for a given logical connection when the connection is made, or ignore that specification and choose its own value for maximum block size. If an application program transmits blocks larger than the downline block size, the network software does not subdivide them until it creates transmission blocks for the terminal.

Application programs should use the downline block sizes defined for terminal devices whenever possible. If the size of an upline or downline network data block is not appropriate for the type of data being exchanged with a device, you should discuss the situation with the network administrator who configures the devices being serviced. The Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface contains guidelines for choosing upline and downline network data block sizes and for selecting terminal transmission block sizes.

Block Limits

Temporary network block storage (queuing) occurs for upline and downline traffic at several points in the network. The network adminstrator controls the storage space required by controlling the network data block size and the number of blocks queued in each direction.

The number of blocks queued depends on several Network Definition Language (NDL) statement parameters. One of those parameters, the ABL parameter, establishes the application block limit. The application block limit is another device configuration parameter received by an application program (as the abl field value) when logical connection occurs. Your application program cannot send more than that number of downline blocks for queuing within the network.

The application block limit can be changed by the application program, using control information called a field number/field value pair; this process is described in section 3. The use of the block limit is described in section 3 as part of the data flow control description.

Transmission Blocks

Terminals send or receive data in physical groupings of character bytes; these groupings are called transmission blocks. The size of a downline transmission block for a specific device is also established by the network administrator (using the XBZ parameter of an NDL statement). The value used might be dictated by hardware requirements.

Transmission blocks exchanged with X.25 devices are called packets and have different size and protocol content requirements than transmission blocks exchanged directly with a terminal. The network administrator can control some of the characteristics of packets.

During upline transmissions from a device, the NPU reassembles the terminal's transmission block into network blocks. Each transmission block from a CDC-defined batch device can contain part of a single message, all of a single message, or several messages. Each transmission block from a CDC-defined console device can contain all of a single message, or several messages.

During downline transmissions, the NPU resassembles network blocks into terminal transmission blocks. This conversion is done so that the application program need not be concerned that output is delivered in appropriately sized transmission blocks when the terminal cannot process blocks larger than a maximum size. Each transmission block can contain part of a single message or all of a single message; downline transmission blocks do not contain more than one message.

INTERACTIVE TERMINAL INPUT CONCEPTS

An interactive device can send or receive data in two modes:

Normalized mode

Transparent mode

The significance of these data modes is described later in this section under Interactive Virtual Terminal Data. The following discussion does not apply to transparent mode data.

In normalized mode, an interactive device transmits logical lines of data. Each logical line is analogous to an English sentence. It is a complete unit of information.

The device can transmit these lines one at a time, or in sets. It therefore can use one of two possible transmission modes.

If the device can transmit only one character or one logical line in each transmission block, it is operating in line mode. If the device can transmit more than one logical line in a transmission block, it is operating in block mode.

HASP and bisynchronous devices (terminal classes 9, 14, 16, and 17) always operate in line mode. X.25 devices (terminal classes 1, 2, and 5 through 8) and mode 4 devices (terminal classes 10 through 13 and 15) always operate in block mode. Only devices in terminal classes 1, 2, and 5 through 8 that do not access the network through an X.25 interface can operate in both modes.

Line Mode Operation

From a terminal user's viewpoint, transmitting a single logical line at a time is a buffered line mode form of input. Buffered line mode allows the user to select either character-by-character or line-by-line transmission (some devices have switches to select either option) without distinction. Each logical line is terminated by an end-of-line indicator; this indicator might also transmit the line from the terminal, if the terminal buffers lines of input. Each logical line becomes a separate network message when the NPU receives it.

When the NPU is told that an interactive device is operating in line mode, the NPU performs line turnaround for it. When a message is sent upline in this mode, the NPU begins to send any downline data available for the device. That is, output is allowed after each logical line of input. (Refer to the KB option for the IN command, described in section 3.)

Block Mode Operation

Some devices can transmit many logical lines in a single transmission block. (The terminal user sometimes can select or override this condition with a BLOCK or BATCH mode switch on the device.) Such devices are called block mode terminals.

Block mode terminals group logical lines in the terminal until the transmission key is pressed; these groups reach the network software as a single transmission block. The network software forwards each message to the application program as a separate transmission; the effect resembles typeahead entries from line mode terminals.

Each logical line within the input transmission block ends with an end-of-line indicator. Each transmission block is terminated by an end-of-block indicator.

Whether each logical line in a transmission block becomes a separate message or each transmission block becomes a single message initially is determined by the network administrator through the device definition in the network configuration file. Your application program or the terminal user can change that mode (refer to the EL and EB options of the EB command, described in section 3).

When the NPU is told an interactive device is operating in block mode, the NPU does not perform line turnaround for it until all of its current transmission block is received. When the terminal is serviced in this mode, the NPU holds all downline data available for the device until it detects the end-of-block indicator. That is, output is allowed after each logical line of input only if each logical line of input is transmitted in a separate block. (Refer to the BK and PT options for the IN command, described in section 3.)

A terminal might have a block transmission key that does not generate the end-of-block indicator. When the block transmission key generates the end-of-line indicator, the terminal is operating in line mode, and logical lines are transmitted from the terminal as separate messages.

When the transmission key does not generate either the currently defined end-of-line indicator or the currently defined end-of-block indicator, the terminal user must be aware of the distinction and must enter an indicator as the last data character before pressing the transmission key. These possible conditions exist:

If the transmission key is pressed immediately after pressing the key that generates an end-of-line indicator, a message is generated. This result is the same as if the device was operating in line mode and the key generating an end-of-line indicator had been pressed, or as if the key generating an end-of-block indicator had been pressed.

If the transmission key is pressed immediately after pressing the key that generates an end-of-block indicator, a message is generated. This result is the same as if the device was operating in line mode and the key generating an end-of-line indicator had been pressed, or as if the transmission key had generated an end-of-block indicator.

If the transmission key is pressed without pressing an end-of-line key or end-of-block key as the last prior activity, an incomplete message exists. The Terminal Interface Program (TIP) generates an upline network data block if enough information was received. If a downline block is available for the device, a few characters of that output might be sent to the device and overwrite some of the input following the last end-of-line or end-of-block indicator received. If downline blocks subsequently become available for the device, the data remains queued while the TIP waits for completion of the input transmission block. This situation exists until the terminal user enters more data, ending with either an end-of-line or an end-of-block indicator.

Physical and Logical Lines

A logical line of input can contain one or more physical lines; a physical line ends when vertical repositioning of the cursor or carriage occurs. If the device recognizes a linefeed operation distinct from a carriage return operation, a physical line ends when a linefeed is entered. If no distinction exists between vertical and horizontal repositioning, a physical line is identical to a logical line.

A physical line of input is relevant to the network software only when a backspace character is processed. Terminal users cannot backspace across physical line boundaries to delete characters in physical lines other than the current one.

A logical line of input always ends when an interactive device transmits an end-of-line or end-of-block indicator. An upline message is normally transmitted to the host as soon as a logical line ends.

End-of-Line Indicators

The end-of-line indicator is initially established by the network administrator when he or she defines the device in the network configuration file. The indicator is either a specific code, a code sequence, or a specific condition associated with use of a certain key or set of keys by the terminal operator. The default keys for generating an end-of-line indicator are shown in table 2-1.

Your application program or the terminal user can change this indicator (refer to the EL command options, described in section 3). The NPU normally discards any end-of-line indicator character code when it detects the end of a logical line.

Multiple Logical Lines in One Message

For upline data from an interactive device, the network administrator can configure the device so that the NPU ignores the character or event that normally causes it to transmit a message as soon as a logical line ends. Instead, he or she can make the NPU use a different character or event to trigger transmission to the host. Your application program or the terminal user can also make this change (refer to the EB option of the EL command, described in section 3).

This option allows the terminal user to pack many logical lines into one set of upline network blocks. Each line includes the end-of-line indicator as a data character that terminates it. This is a form of single-message mode, because the host receives only one message. From the terminal user's viewpoint, one message is many logical lines.

End-of-Block Indicators

The end-of-block indicator is initially established for the device by the network administrator when he or she defines the device in the network configuration file. The indicator is either a specific code, a code sequence, or a specific condition associated with use of a certain key or set of keys by the terminal operator.

The default keys for generating an end-of-block indicator are shown in table 2-1. In X.25 packet-switching networks, the packet transmission condition is always the end-of-block indicator.

When the device is not operating in block mode, the end-of-block indicator has the same effect as an end-of-line indicator.

Your application program or the terminal user can change the end-of-block indicator (refer to the EB command, described in section 3). This indicator normally is discarded when the last message from the device is sent upline.

TABLE 2-1. DEFAULT MESSAGE DELIMITER AND TRANSMISSION KEYS

Terminal Class	Archetype Terminal	End-of-Line Key	Character or Line Mode Transmission Key	Block Mode Transmission Key
1	Teletype Model 30 series	RETURN	RETURN	CTRL and D
2	CDC 713, 751, 756	RETURN or CARRIAGE RETURN	RETURN or CARRIAGE RETURN	SEND or CONTROL and D
3	Reserved for CDC use			
4	IBM 2741	RETURN	RETURN	None
5	Teletype Model 40-2	RETURN	RETURN	SEND
6	Hazeltine 2000	CR	CR	SHIFT and XMIT or CTRL and D
7	CDC 752	CARRIAGE RETURN	CARRIAGE RETURN	CTRL and D
8	Tektronix 4014	RETURN	RETURN	CTRL and D
1, 2, 5 thru 8	X.25 packet assembly/ disassembly (PAD) console device	RETURN	Packet transmission key	Packet transmission key
9	HASP (postprint)	Variable	Variable	None
10	CDC 200 User Terminal	RETURN	None	SEND
11	CDC 714-30	NEW LINE	None	ETX
12	CDC 711	NEW LINE	None	ETX
13	CDC 714-10/20	NEW LINE	None	ETX
14	HASP (preprint)	Variable	Variable	None
15	CDC 734	NEW LINE	None	SEND
16	IBM 2780	End of card	End of card	None
17	IBM 3780	End of card	End of card	None
18 thru 28	Reserved for CDC use			
29 thru 31	Site-defined	Unknown	Unknown	Unknown

INTERACTIVE TERMINAL OUTPUT CONCEPTS

A downline message can contain many logical lines of output. Each logical line can contain many physical lines of output.

A logical line of output ends when the application program embeds a code or set of bytes for that purpose in the message, or when the block containing the line ends. A downline message ends when an application program indicates that condition.

Because downline messages can always contain more than one logical line, an interactive device can always receive the output equivalent of a multiple-message block mode input transmission. The application program can group logical lines as necessary to achieve that effect.

If a message fits into a downline network data block, the block becomes a single-block message. If one downline message cannot be fit into a single network data block, the application program can split it into as many blocks as necessary. An application program generally sends a single message (consisting of as many logical lines as necessary) as the response to one input message from an interactive device.

BATCH DATA

Batch devices can be serviced through the interactive virtual terminal interface described later in this section. A separate set of interface protocols also exists for batch devices serviced by CDC-written Terminal Interface Programs and application programs.

These programs require large amounts of data to be exchanged between a host computer's mass storage devices and CDC-defined batch devices. Such batch data is therefore assembled into messages of one or more network data blocks. Each network data block contains one or more mass storage physical record units (PRUs). Because only the CDC-written Remote Batch Facility can use the special interface for CDC-defined batch devices, the remainder of this manual does not discuss the requirements this interface imposes on batch data or batch device support.

INFORMATION IDENTIFICATION PROTOCOLS

Conventions exist for identifying network blocks. These conventions indicate the following things to the application program sending or receiving the block:

The kind of message of which the block is a part; this is called the message type.

The kind of information within the block; this is called the application block type.

The areas of host central memory containing the block and containing information describing the block; these are called the block buffer areas.

The source or destination of the block; these are called the application connection number and the application list number.

APPLICATION PROGRAM MESSAGE TYPES

An application program message is a complete logical unit of information, comprising one or more physical network blocks. A message can be a line of data to or from a teletypewriter, a mass storage file, a service request to NAM, or a screen of information for a cathode ray tube.

There are two kinds of application messages, data and supervisory. Data messages convey information of significance only to a device user or to another application program. Data messages can consist of more than one network data block.

Supervisory messages convey information of significance only to the network software. Supervisory messages consist of only one network block.

Supervisory messages are used by an application program to control data messages between itself and logical connections.

APPLICATION BLOCK TYPES

The network block is the basic unit of information exchange for the application program. There are several types of network blocks that an application program can exchange. Each type has an identifying application block type number assigned to it. The following types exist:

Null blocks, which are dummy input blocks indicating the absence of any data or supervisory information. These blocks have an application block type number of 0.

Blocks containing portions of data messages, but not terminating those messages. These blocks have an application block type number of 1; such blocks are called BLK blocks in other network documentation.

Blocks that terminate data messages. These blocks can include physically empty blocks when such blocks convey logical information. Blocks that terminate data messages have an application block type number of 2; such blocks are called MSG blocks in other network documentation.

Blocks constituting supervisory messages. These blocks have an application block type number of 3; such blocks include the information in blocks called CMD, BACK, BRK, NAK, and other acronyms in some network documentation.

BLOCK BUFFER AREAS

All network blocks are exchanged between the application program and the network software using two kinds of buffers:

The block header area

The block text area

Block Header Area

Block header areas each contain a 60-bit word describing the contents of a corresponding text area. This block header word accompanies the block in the corresponding block text area during the exchange between the application program and NAM.

For downline blocks, the application program creates the block header and NAM interprets it. For upline blocks, NAM creates the block header and the application program interprets it.

Because the contents of the header word depend on the contents of the text area, the header word formats are described in this manual after the text area content protocols are described. To simplify

the header area descriptions, they are presented in four separate formats:

For upline network data blocks

For downline network data blocks

For upline supervisory message blocks

For downline supervisory message blocks

Block Text Area

A block text area is separately addressed from its header area and need not be contiguous to it. The text area contains the single network block described by the header word in the header area.

Text areas can be of varying length, as necessary to accommodate various block lengths. The text area has a maximum length expressed as a whole number of central memory words. Text areas can be up to 410 central memory words long.

The length of the text area used by the application program is described to the network by the application program. The text area length must be calculated from the maximum length of the blocks it will contain.

Block length is distinct from text area length. The length of a block depends on the type and use of the block.

Null blocks have zero length and do not require any central memory words for their text area. Other block types have lengths expressed in character byte units, although the bytes need not actually contain characters.

Blocks are always a whole number of character units long but do not have to be a whole number of central memory words long. Not all words in the text area used for a given block need to be filled with meaningful information.

Supervisory message blocks are 1 to 410 words long. Data blocks have lengths of zero up to the maximum number of characters that can fit in the maximum text area of 410 words, or 2043 characters, whichever occurs first.

Downline messages containing more characters than the text area size can hold must be divided into several network data blocks. Each such block must fit into the text area. Each of these blocks should also meet the network block size requirement and must be transmitted separately.

CONNECTION IDENTIFIERS

Two parameters identify and control the routing of messages:

The application connection number

The application list number

Both parameters are used in AIP calls that fetch incoming network data blocks. The application connection number is used in the block header words of outgoing blocks.

Application Connection Number

The application connection number is a 12-bit integer used to address a particular logical connection. The connection number can be used as an index into a control structure (for example, the number of a connection could be the ordinal of a corresponding device table) or used in any other manner the application chooses.

These connection numbers are assigned serially by NAM for each application program. Numbers that become available because of disconnections are reassigned to subsequent connections.

A connection number of zero indicates the control connection on which asynchronous supervisory messages are sent and received. (See Supervisory Message Content and Sequence Protocols, later in this section.)

Application List Number

NAM permits an application program to group connections with similar processing requirements into numbered lists. This is an efficiency feature, relieving the application of the need to specify individual connections each time upline block processing is required. Instead, when a request is made for a block from a connection on a list, any device or application program connections with empty input queues are automatically skipped and a block from the first nonempty queue is returned. A single null block is returned when none of the connections on the list have any input queued.

This feature can be used in many kinds of list structures. For example:

An application program must process input from devices with large network block sizes (such as interactive graphics terminals in a specific terminal class) differently than input from devices with small block sizes. This processing occurs in different portions of the program code; therefore, the application program assigns the devices using large blocks to list 1 and the devices using small blocks to list 2.

An application program treats all devices the same and must process blocks from them on an equal basis. Accordingly, it assigns them all to the same list.

An application program services terminals in four geographical areas; each must be treated separately because of varying state laws. Accordingly, they are assigned to lists 1 through 4.

An application program services devices that should be treated the same, but with the following complication: when the application has received a block from a particular terminal, it must perform some time-consuming function that prevents it from immediately processing another block from the same terminal. Accordingly, the application places all connections on list 1 and issues an input request on list 1. When a block for connection x is returned, it temporarily inhibits receipt of data on connection x before it issues the next input request. When it can accept another data block from the terminal using logical connection x, the application program sends a supervisory message to reverse the effect of the temporary inhibition.

The parameter used for this kind of processing is called the application list number. The application list number is an integer from 0 through 63 specified by the application program when it accepts a connection. NAM links message input (upline) queues of all connections that have been assigned the same list number. An application program can request blocks from these linked queues in rotation (without specifying individual connections) by including the assigned application list number in a NETGETL or NETGTFL statement (described in section 5).

Each list number identifies one connection list. A connection list can be viewed as a table of connection numbers. These connection numbers are entered in the table in the order in which the application program assigns the connections to the list. When the list is scanned for input from a connection, the connections are examined in the order in which they are entered in the table.

The application program explicitly assigns the list number to each logical connection when the connection is established. The logical connection corresponding to application connection number zero already exists when the application is connected to the network. For this reason, application connection number zero is automatically assigned to application list number zero without program intervention.

The application program does not have to maintain any tables associating connection numbers and list numbers. The application program need not use list processing at all.

DATA MESSAGE CONTENT AND SEQUENCE PROTOCOLS

Data blocks consist of 1 to 410 60-bit words or 1 to 2043 12-bit bytes. The fields within these blocks convey information to or from the terminal user. Data blocks have associated block header words. These header words convey information to the network software concerning the contents of the corresponding text area buffer.

Data blocks are sent and received through the Application Interface Program routines described in section 5. The application program fetches data messages one block at a time. When the connection queue is empty, a null block with an application block type of zero is returned.

The network software provides a mechanism for the application program to determine when data blocks are queued. When a call to an AIP routine is completed, a supervisory status word at a location defined by the application program is updated to indicate whether any data blocks are queued. As long as the application program continues to make calls to AIP routines, it can test the supervisory status word periodically (instead of attempting to fetch null blocks from all application connection numbers). The supervisory status word and the use of NETWAIT are described in section 5.

The protocols for data message text and the use of the text area buffer depend on whether the logical connection is with another application program, an interactive virtual terminal device, or a passive batch device. Blocks exchanged with other application programs in the same host have the fewest requirements and most flexible structure. Blocks exchanged with CDC-defined batch devices using the special batch device protocol have the most requirements and the least flexible structure.

Requirements for blocks exchanged with other application programs in the same host are covered in the figures later in this section, and in section 3. Blocks exchanged between application programs are groups of binary character bytes with no parity, equivalent to transparent mode data. Such blocks can use the eighth bit of an 8-bit byte as data and need not have the transparent mode bit set in their block header; see the decriptions of transparent mode and block header word content later in this section.

The requirements for exchanging blocks with interactive virtual terminal devices are described below. Requirements for blocks exchanged with batch devices through the special batch device interface are not described because that interface is available only to RBF.

INTERACTIVE VIRTUAL TERMINAL DATA

An interactive virtual terminal can be either a CDC-defined console device or a site-defined device. An interactive virtual terminal can send and receive data in two modes: normalized mode and transparent mode. The format and content of data in these modes is described later in this subsection. The characteristics of an interactive virtual terminal depend on which data exchange mode is currently used.

In normalized mode, the characteristics of an interactive virtual terminal are as follows:

Input and output can occur simultaneously.

A page of output has infinite (no physical) width; logical lines are divided automatically as needed to fit the physical line restrictions of the device.

A page of output has infinite (no physical) length; sets of logical lines are divided automatically as needed to fit the physical restrictions of the device page.

A logical line of output cannot be longer than a single network block; a single message can contain an infinite number of logical lines.

Characters are either 7-bit ASCII codes using zero parity (bit 7, the eighth bit, is always zero in upline data and ignored in downline data), or 6-bit display codes with no parity.

Logical lines of input are terminated by a changeable character or condition; this terminator is the end-of-line or end-of-block indicator described earlier in this section. The input terminator is not part of the data seen by an application program unless the full-ASCII feature is used (this is explained later in this subsection and in section 3 where the FA command is described).

Logical lines of output are terminated by an ASCII unit separator character code (US, represented by the hexadecimal value 1F) or the end of a zero-byte terminated record. The application program places this terminator in the data.

No cursor positioning actions are required to acknowledge receipt of input, and no timing adjustments need to be made at the end of physical output lines.

Logical lines can be divided into physical lines by embedding optional format control characters in downline blocks.

In transparent mode, the characteristics of an interactive virtual terminal are as follows:

Input and output can occur simultaneously.

A page of output has infinite (no physical) width.

A page of output has infinite (no physical) length.

Characters are either 7-bit codes using zero parity (bit 7, the eighth bit, is always zero in upline data and ignored in downline data), or codes of a terminal-dependent code set with terminal-dependent parity.

Messages of input are terminated by a changeable character or condition; this terminator is one of the message or mode delimiters described later in this section. The mode delimiter is not part of the data seen by an application program.

Messages of output are terminated by a condition or event chosen by an application program (each network block is separately designated as transparent or normalized when sent).

Cursor positioning actions might be required, and timing adjustments might need to be made at the end of physical output lines.

Line Turnaround Convention

The interactive virtual terminal concept imposes some conventions on the content and sequencing of blocks exchanged with an interactive device. The primary convention of block sequencing involves the direction and time of block transmission.

The application program can service an interactive device on a connection as if the device always operates in a full-duplex mode. That is, input and output can occur independently; the terminal user can enter several logical lines at once (an operation called typeahead), without waiting for a response to each line.

Application program input and output need not alternate. However, some devices cannot actually operate that way. To prevent a loss of synchronization between input and output at such devices, a line turnaround convention exists. This convention consists of the following events.

After a block of type 2 (the end of a message) is sent to a device, no more blocks should be sent downline until at least one block is input from the same device. An application program therefore should never send the last block of a message downline until it is ready to wait for input.

A network data block of type 2 has special significance to the network software during output to an interactive device. When such a block is the last block of the output stream, the network software:

Unlocks the keyboard of an interactive device being serviced as terminal class 4 (an IBM 2741).

Sends an X-ON code to start an automatic paper tape input mechanism, if one has been defined as the input mechanism for the device. Paper tape operation is explained in more detail in section 3 where the IN and OP commands are described.

Starts polling devices in terminal classes 10 through 13 and 15 (mode 4 consoles).

Identifies an automatic input prompt to be returned, if the application program uses this feature. When this feature is used, the network software delivers the block to the device and retains the first 20 characters in the NPU's input buffer. Subsequent input from the device is attached to the end of the retained data. (If more than one logical line is received from the device, the first is appended to the retained data.) All logical lines are transmitted to the host as received from the device.

If the terminal is a half-duplex device, such as a 2741 or a paper tape reader/punch, it must enter input before the network software will deliver additional output messages. Other devices are not subject to this restriction.

The requirement for an input block after a block of type 2 is output can be satisfied in several ways by terminal operators. An empty input line can be entered and will reach the application program as a block of type 2 but containing nothing. A line containing data can be entered and will reach the application program as one or more network data blocks.

Devices can interrupt output by entering input. When this occurs, the network software stops the output until the terminal user completes the input (using an end-of-line or end-of-block indicator). Output then resumes at the next character of the current physical and logical line.

INTERACTIVE VIRTUAL TERMINAL EXCHANGE MODES

The conventions of block content depend on the mode in which the block is exchanged. There are two possible exchange modes, normalized mode and transparent mode. The latter is referred to in other documentation as binary mode. This manual uses transparent mode to indicate exchange of a block that is not in normalized mode.

Normalized Mode Operation

The interactive virtual terminal interface assembles message character streams into upline network data blocks from terminal transmission blocks. It disassembles character streams from downline network data blocks, reassembling them into terminal transmission blocks.

The assembly operation is controlled by the termination of logical lines. The disassembly operation can be controlled by the termination of physical lines when that is appropriate for the output mechanism of the device. The disassembly operation can also be modified by format control characters embedded in each block, and by the page width defined for the device (refer to the PW command in section 3).

End of Logical Lines in Input

Logical lines reach an application program as one or more network data blocks. Logical lines usually end when a message ends and do not contain the character or code sequence defined as the end-of-line or end-of-block key.

However, two special cases exist. Logical lines do contain the end-of-line or end-of-block codes when the device is operating in full-ASCII editing mode (described later in this section). Logical lines also contain the end-of-line code when the end-of-line key is changed to be the default end-of-block key for the device (see the EB option of the EL command described in section 3). In the latter case, the transmission block becomes a message, and the logical lines within it have no effect on construction or type of network data blocks.

Logical and Physical Lines in Output

The application program does not need to equate a logical line of output to a complete message nor does it need to create a separate network block for each physical line of output. A single logical line can contain many complete physical lines. A single block can contain many complete logical lines, and a message can be one or many such blocks. A physical or logical line cannot, however, be continued from one block to another.

Logical lines within downline blocks are ended by an end-of-line indicator. Unlike the end-of-line indicators used in upline blocks, downline blocks always contain codes for the end-of-line function; the codes used downline are always the same and usually differ from the codes used upline. The downline end-of-line indicator varies according to the application character type of the block; application character types are described later in this section. Bytes used to store indicators must be included when determining the number of characters comprising a downline block.

The end-of-line indicator in 60-bit character bytes (application character type 1) is determined by the programs exchanging the block. No predefined end-of-line indicator exists for that application character type.

The end-of-line indicator in blocks using 8-bit characters in 8-bit or 12-bit bytes (application character types 2 or 3) is determined by whether the block is sent in normalized mode or transparent mode (described later in this section). In transparent mode, no end-of-line indicator exists. In normalized mode, the end-of-line indicator is the ASCII unit separator character US.

The end-of-line indicator in blocks using 6-bit character bytes (application character type 4) is 12 to 66 bits of zero; these bits are right-justified to fill the last central memory word involved. This convention makes each logical line the equivalent of a zero-byte terminated logical record.

The 6-bit option requires a right-justified 12-bit byte in at least one central memory word. On computers using the 64-character set, the colon is represented in 6-bit display code by six zero bits. On such systems, if the application needs to send colons to the terminal console in 6-bit display code, care must be taken to make sure that a string of colons is not interpreted as an end-of-line indicator. A colon preceding the end-of-line indicator is considered as part of the indicator and not as a colon when it occupies one of the two rightmost character positions in the next-to-last central memory word of the block or any of the eight leftmost positions in the last word of the block.

All predefined end-of-line indicators embedded within a block are discarded by the network software and produce no characters on the console output device. The network software can perform carriage or cursor repositioning when an end-of-line indicator is encountered; this operation is described later in this section under Format Effectors.

Upline Character Sets and Editing Modes

The network protocol permits entry from a device of any code less than or equal to 8 bits per character; however, a normalized mode character always reaches an application program as one of the 128 ASCII characters defined in appendix A. Receipt of an entered character by the application program depends on the editing functions performed by the TIP. Three editing modes exist for the TIP when it processes normalized data:

Complete interactive virtual terminal editing mode

Special editing mode

Full-ASCII mode

Devices always begin a connection with the network in normalized mode. The initial upline editing mode is established for each device when the device is defined by the network adminstrator. The application program or the terminal user can change that mode using the SE or FA commands, described in section 3.

Complete Editing

During complete editing operations, the following hexadecimal character codes cannot be received by the network application program:

00 (the ASCII character NUL)

7F (the ASCII character DEL)

0A (the ASCII character LF)

The backspace character code currently defined for the device (see the BS command in section 3)

The end-of-line character currently defined for the device (see the EL command in section 3)

The end-of-block character currently defined for the device (see the EB command in section 3)

The following hexadecimal character codes cannot be received, if entered at certain points in a message:

11 (the ASCII character DC1) if it follows an end-of-line or end-of-block character and the TIP is supporting output control for the device (see the Y option of the OC command in section 3)

13 (the ASCII character DC3) if it follows an end-of-line or end-of-block character and the TIP is supporting output control for the device (see the Y option of the OC command in section 3).

13 (the ASCII character DC3) if it follows an end-of-line or end-of-block character and the input mechanism is known to be a paper tape reader (see the PT option of the IN command in section 3)

02 (the ASCII character STX), if entered as the first character of a message

The user-break-1 and user-break-2 character codes currently defined for the terminal, if entered as the only character in a message (see the B1 and B2 commands in section 3)

The abort-output-block character code currently defined for the terminal, if entered as the only character in a message (see the AB command in section 3)

The network control character currently defined for the terminal when it follows an end-of-line or end-of-block character or when it is used for such purposes as page turning (see the CT command and the Y option of the PG command in section 3)

The currently defined cancel input character is always received at the end of the logical line it cancels. This character is not data.

Special Editing

Special editing takes precedence over complete editing. Special editing cannot occur if the terminal operates in block mode.

When special editing occurs, linefeed codes and the currently defined backspace code are forwarded to the application program as data. The network software sends appropriate responses to the device when it receives these codes.

During special editing operations, the following hexadecimal character codes cannot be received by the network application program:

00 (the ASCII character NUL)

7F (the ASCII character DEL)

The end-of-line character currently defined for the device (see the EL command in section 3)

the end-of-block character currently defined for the device (see the EB command in section 3)

The following hexadecimal character codes cannot be received, if entered at certain points in a message:

11 (the ASCII character DC1) if it follows an end-of-line or end-of-block character and the TIP is supporting output control for the device (see the Y option of the OC command in section 3)

13 (the ASCII character DC3) if it follows an end-of-line or end-of-block character and the TIP is supporting output control for the device (see the Y option of the OC command in section 3).

13 (the ASCII character DC3) if it follows an end-of-line or end-of-block character and the input mechanism is known to be a paper tape reader (see the PT option of the IN command in section 3)

02 (the ASCII character STX), if entered as the first character of a message

The user-break-1 and user-break-2 character codes currently defined for the terminal, if entered as the only character in a message (see the B1 and B2 commands in section 3)

The abort-output-block character code currently defined for the terminal, if entered as the only character in a message (see the AB command in section 3)

The network control character currently defined for the terminal when it follows an end-of-line or end-of-block character or when it is used for such purposes as page turning (see the CT command and the Y option of the PG command in section 3)

The currently defined cancel input character is always received at the end of the logical line it cancels. This character is not data.

Full-ASCII Editing

Full-ASCII editing takes precedence over special editing or complete editing. When full-ASCII editing occurs, almost all codes are forwarded to the application program as data. The network software does not perform actions at the terminal when it receives the codes for backspace, abort-output-block, cancel input message, user-break-1, or user-break-2. These codes and the end-of-line and end-of-block indicator codes are sent upline as data.

During full-ASCII editing operations, the following hexadecimal character codes cannot be received by the network application program:

00 (the ASCII character NUL) if it occurs after the end-of-line or end-of-block indicator

0A (the ASCII character LF) if it occurs after the end-of-line or end-of-block indicator

7F (the ASCII character DEL) if it occurs after the end-of-line or end-of-block indicator

The network control character currently defined for the terminal if it occurs after the end-of-line or end-of-block indicator or when it is used for such purposes as page turning (see the CT command and the Y option of the PG command in section 3)

11 (the ASCII character DC1) if it occurs after the end-of-line or end-of-block indicator and the TIP is supporting output control for the device (see the Y option of the OC command in section 3)

13 (the ASCII character DC3) if it occurs after the end-of-line or end-of-block indicator and the TIP is supporting output control for the device or is explicitly supporting paper tape input from the device (see the Y option of the OC command and the PT option of the IN command in section 3).

The currently defined cancel input character is always received as the last character of the logical line it ended. This character is data.

Downline Character Sets

The network protocol permits transmission from a network application program of any character code less than or equal to 8 bits. If the application program uses one of the application character types that permits transmitting an 8-bit code (application character types 2 and 3), it cannot use the upper (eighth) bit for data unless it is transmitting in transparent mode.

In normalized mode, the application program can only use the 128 ASCII characters defined in appendix A. If the application program transmits a 7-bit ASCII code, it cannot use the upper (eighth) bit for parity; the network ignores the eighth bit in downline normalized mode data.

Receipt of a transmitted character by the device depends on the editing functions and character transformations performed by the TIP. In addition to character codes altered during the translation and substitution operations described elsewhere in this section and in appendix A, the hexadecimal character code 1F (the ASCII character US used as a downline block end-of-line indicator) cannot be received by a device when the application program transmits a block in normalized mode.

Page Width and Page Length

The application program receives an indication of the page width and page length in effect for a device when connection with the device first occurs. The application program or the terminal user can change the page width and page length in effect for a device.

The Terminal Interface Program uses the page length defined for the device to format physical lines into physical pages or screens of output. The Terminal Interface Program uses the page width value to transform logical lines of downline data into physical lines of output.

For console devices defined as having hardcopy output mechanisms (see the PR option of the OP command in section 3), a logical line of downline data containing more characters than the page width value permits is divided into singly spaced physical lines. These physical lines are equal to or shorter than the page width in effect and are displayed successively.

For all console devices, the page width is used as part of the line-counting algorithm to determine the page length. Each logical line is examined to determine how many multiples of the page width (how many physical lines) it contains. Each complete or partial multiple counts as one line when the TIP determines the page length.

Line counting begins at the beginning of each downline message. The line counter is reset to zero each time the page length of the terminal is reached, each time any input occurs, or when page turning occurs during page waiting operation. Refer to the PG, PW, and PL commands in section 3.

The physical line width of the device might be smaller than the page width defined for the device. When this happens, the effect of sending a logical line of downline data containing more characters than the physical line width permits depends on the terminal hardware.

Format Effectors

An application program can control the presentation of the characters within a data block by indicating that the block contains format effectors. If the application program chooses to do this, the first character of each logical line within the block becomes a format effector. Format effector characters cause predefined formatting operations when the block is delivered to the device. The network software discards these characters after interpretation; therefore, these characters do not appear on the interactive terminal output device.

You must include format effector characters when determining the number of characters comprising the block. Format effector characters are excluded from page width calculations.

Tables 2-2 and 2-3 describe the predefined operations produced by each format effector character of each terminal class. The Terminal Interface Program performs the predefined format effector operation by inserting the codes for the characters indicated in the tables in place of the discarded format effector character code. The inserted terminal codes are those of characters in the ASCII set described in appendix A, with the exception that NL indicates the terminal-defined new-line code sequence.

TABLE 2-2. FORMAT EFFECTOR OPERATIONS FOR ASYNCHRONOUS AND X.25 CONSOLES

Terminal Class	Format Effector	General Physical Operation	Is Infinite Page Length Declared?	Does Output Follow Previous Input	Code Substituted on Output Mechanism†	
					Display or Printer	Paper Tape
1	blank	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
	0	Space 2 lines before output.	Does not matter	Yes No	CR, LF CR, 2LF	CR, LF CR, 2LF
	-	Space 3 lines before output.	Does not matter	Yes No	CR, 2LF CR, 3LF	CR, 2LF CR, 3LF
	+	Position to start of current line before output.	Does not matter	Yes or No	CR	CR
	*	Position to top of form or home cursor before output.	Yes	Yes No	CR, 5LF CR, 6LF	CR, 5LF CR, 6LF
			No	Yes or No	Calculated by TIP	
	1	Position to top of form or home cursor and clear screen before output.	Yes	Yes No	CR, LF CR, 6LF	CR, 5LF CR, 6LF
			No	Yes or No	Calculated by TIP	
	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	CR,LF	CR,LF, DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 2NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
2	blank	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
	0	Space 2 lines before output.	Does not matter	Yes No	CR, LF CR, 2LF	CR, LF CR, 2LF
	-	Space 3 lines before output.	Does not matter	Yes No	CR, 2LF CR, 3LF	CR, 2LF CR, 3LF
	+	Position to start of current line before output.	Does not matter	Yes or No	CR	CR
	*	Position to top of form or home cursor before output.	Does not matter	Yes or No	EM	EM
	1	Position to top of form or home cursor and clear screen before output.	Does not matter	Yes or No	EM, CAN	EM, CAN

TABLE 2-2. FORMAT EFFECTOR OPERATIONS FOR ASYNCHRONOUS AND X.25 CONSOLES (Contd)

Terminal Class	Format Effector	General Physical Operation	Is Infinite Page Length Declared?	Does Output Follow Previous Input	Code Substituted on Output Mechanism†	
					Display or Printer	Paper Tape
2 (Contd)	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	CR, LF	CR, LF DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 2NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
4††	blank	Space 1 line before output.	Does not matter	Yes No	None NL	N/A
	0	Space 2 lines before output.	Does not matter	Yes No	NL 2NL	N/A
	-	Space 3 lines before output.	Does not matter	Yes No	2NL 3NL	N/A
	+	Position to start of current line before output.	Does not matter	Yes or No	nBS n is calculated by TIP from current position	N/A
	*	Position to top of form or home cursor before output.	Yes	Yes No	5NL 6NL	N/A
			No	Yes or No	nNL n is calculated by TIP from current position	N/A
	1	Position to top of form or home cursor and clear screen before output.	Yes	Yes No	5NL 6NL	N/A
			No	Yes or No	nNL n is calculated by TIP from current position	N/A
	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	NL	NL
	/	Position to start of current line after output.	Does not matter	Yes or No	nBS n is calculated by TIP from current position	nBS
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	None NL	None NL

TABLE 2-2. FORMAT EFFECTOR OPERATIONS FOR ASYNCHRONOUS AND X.25 CONSOLES (Contd)

Terminal Class	Format Effector	General Physical Operation	Is Infinite Page Length Declared?	Does Output Follow Previous Input	Code Substituted on Output Mechanism†	
					Display or Printer	Paper Tape
5	blank	Space 1 line before output.	Does not matter	Yes No	None LF	None LF
	0	Space 2 lines before output.	Does not matter	Yes No	LF 2LF	LF 2LF
	-	Space 3 lines before output.	Does not matter	Yes No	2LF 3LF	2LF 3LF
	+	Position to start of current line before output.	Does not matter	Yes or No	ESC, G	ESC, G
	*	Position to top of form or home cursor before output.	Does not matter	Yes or No	ESC, H	ESC, H
	1	Position to top of form or home cursor and clear screen before output.	Does not matter	Yes or No	ESC, R	ESC, R
	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	LF	LF, DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	ESC, G	ESC, G, DC3, 2NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	None LF	None LF
6	blank	Space 1 line before output.	Does not matter	Yes or No	CR	CR
	0	Space 2 lines before output.	Does not matter	Yes No	CR 2CR	CR 2CR
	-	Space 3 lines before output.	Does not matter	Yes No	2CR 3CR	2CR 3CR
	+	Position to start of current line before output.	Does not matter	Yes or No	None	None
	*	Position to top of form or home cursor before output.	Does not matter	Yes or No	DC2	DC2
	1	Position to top of form or home cursor and clear screen before output.	Does not matter	Yes or No	FS	FS
	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	CR	CR, DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	None	DC3, 2NUL

TABLE 2-2. FORMAT EFFECTOR OPERATIONS FOR ASYNCHRONOUS AND X.25 CONSOLES (Contd)

Terminal Class	Format Effector	General Physical Operation	Is Infinite Page Length Declared?	Does Output Follow Previous Input	Code Substituted on Output Mechanism†	
					Display or Printer	Paper Tape
6 (Contd)	Any other ASCII character	Space 1 line before output.	Does not matter	Yes or No	CR	CR
7	blank	Space 1 line before output.	Does not matter	Yes No	CR CR,LF	CR CR, LF
	0	Space 2 lines before output.	Does not matter	Yes No	CR, LF CR, 2LF	CR, LF CR, 2LF
	-	Space 3 lines before output.	Does not matter	Yes No	CR, 2LF CR, 3LF	CR, 2LF CR, 3LF
	+	Position to start of current line before output.	Does not matter	Yes or No	CR	CR
	*	Position to top of form or home cursor before output.	Does not matter	Yes or No	EM	EM
	1	Position to top of form or home cursor and clear screen before output; delay 100 millisecends before further output.	Does not matter	Yes or No	EM, CAN	EM, CAN
	,	Do not change position before output.	Does not matter	Yes or No	None	None
	.	Space 1 line after output.	Does not matter	Yes or No	CR, LF	CR, LF DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 2NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
8	blank	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
	0	Space 2 lines before output.	Does not matter	Yes No	CR, LF CR, 2LF	CR, LF CR, 2LF
	-	Space 3 lines before output.	Does not matter	Yes No	CR, 2LF CR, 3LF	CR, 2LF CR, 3LF
	+	Position to start of current line before output.	Does not matter	Yes or No	CR	CR
	*	Position to top of form or home cursor before output.	Does not matter	Yes or No	ESC, FF	ESC, FF
	1	Position to top of form or home cursor and clear screen before output; delay 1 second before further output.	Does not matter	Yes or No	ESC, FF	ESC, FF
	,	Do not change position before output.	Does not matter	Yes or No	None	None

TABLE 2-2. FORMAT EFFECTOR OPERATIONS FOR ASYNCHRONOUS AND X.25 CONSOLES (Contd)

Terminal Class	Format Effector	General Physical Operation	Is Infinite Page Length Declared?	Does Output Follow Previous Input	Code Substituted on Output Mechanism†	
					Display or Printer	Paper Tape
8 (Contd)	.	Space 1 line after output.	Does not matter	Yes or No	CR, LF	CR, LF, DC3, 2NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 2NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF

†Paper tape column does not apply to X.25 devices.

††X.25 devices cannot belong to terminal class 4.

TABLE 2-3. FORMAT EFFECTOR OPERATIONS FOR SYNCHRONOUS CONSOLES

Terminal Class	Format Effector	General Physical Operation†	
		Before Output	After Output
9 and 14	0	Space 1 line.	Space 1 line.
	-	Space 2 lines.	Space 1 line.
	Any other ASCII character	None.	Space 1 line.
10 thru 13 and 15	blank	None.	Space 1 line.
	0	Space 1 line.	Space 1 line.
	-	Space 2 lines.	Space 1 line.
	*	Position to top of form or home cursor.	Space 1 line.
	1	Position to top of form or home cursor and clear screen.	Space 1 line.
	Any other ASCII character	None.	Space 1 line.
16 and 17	Any ASCII character	Before the first line of the message, generate the prefix text ***CONSOLE MESSAGE	Space 1 line.
		Before the subsequent lines of the message, do nothing.	Space 1 line.

†No direct correspondence to code substituted on output device can be made. Code used for implementation depends on placement of message blocks within a transmission.

Numbers preceding codes indicate the number of times the codes are repeated in the inserted sequence. Each line output to a console in terminal classes 9 through 17 leaves the cursor positioned at the beginning of the next physical line. Processing of the next line takes this into account.

The format effector characters for clear screen and home cursor operations (* and 1) receive special treatment by the Terminal Interface Program when it is performing a page wait function for the terminal. (See the PG command in section 3.) If these characters are encountered when the TIP has output only part of a page, the TIP pauses for terminal operator acknowledgment of the partial page. When acknowledgment occurs, the format effector functions are performed and output continues automatically. This pause occurs without application program action or knowledge.

If the application program does not indicate the existence of format effectors, the first character of each logical line does not act as a format effector. These characters are output normally but are preceded by the character codes necessary to space one line before output. These default line-spacing codes are the ones substituted when a blank is used as a format effector.

The application program sets a field in the downline block's header word to indicate whether the block contains format effectors. This indication, however, has no effect on the use of format control characters within logical lines of the block. Table 2-4 lists the code substitutions performed for embedded control characters during output to a device in each terminal class. This table uses the same character representation convention as tables 2-2 and 2-3, with the following exceptions: the hexadecimal terminal codes are shown for multiple ASCII character sequences or for non-ASCII character sequences.

Transparent Mode Operation

Blocks exchanged between an application program and a console device in transparent mode do not use most of the features of the interactive virtual terminal interface:

No input editing occurs.

No code conversion occurs.

No format effector transformations are performed for downline blocks.

No page width operations are performed to preserve physical line boundaries.

Page waiting occurs only at the end of a downline message.

Transparent mode operation is separately selected for input and output. Either the terminal operator or the application program can start transparent mode input, using the IN command described in section 3. Only the application program can start transparent mode output.

Data blocks input in transparent mode have a field set in their associated header word to indicate this condition. Output blocks require the same field to be set.

Transparent mode data exchanged with terminal devices can occupy up to 8 bits of an 8-bit byte, representing up to 256 distinct character codes of device instructions. Codes longer than 8 bits cannot be exchanged; data packed in 12-bit bytes by an application program or a terminal device is truncated to 8 bits by the network software.

HASP terminals (terminal classes 9 and 14) and bisynchronous terminals (terminal classes 16 and 17) cannot transmit or receive such blocks. All other terminals can, although mode 4 terminals (terminal classes 10 through 13 and 15) require the special treatment described below.

During transparent mode operation, the application program is responsible for all data formatting and terminal control. For mode 4 terminals, this means that the Terminal Interface Program does not blank-fill the current line and unlock the keyboard before input can be performed but does add or remove the line transmission portion of the protocol envelope to or from all message text exchanged with the terminal.

Two mutually exclusive forms of transparent mode input can be selected. The network administrator can make this selection when the device is defined in the network configuration file, or the application program or the terminal operator can make it while the device is active. The two forms are:

Single message

Multiple message (analogous to block mode operation)

Single-Message Input

For single-message input, one or more transparent mode input delimiters are specified, using the DL command options described in section 3. For single-message input, a message ends when transparent mode input ends. Transparent mode messages need not be equivalent to normalized mode logical lines.

Single-message transparent mode input ends when the Terminal Interface Program encounters one of the mode delimiter conditions. The delimiter conditions are:

Occurrence of a specific character code in the input

Occurrence of a specific number of character bytes in the input

Occurrence of a 200- to 400-millisecond timeout in the input

TABLE 2-4. EMBEDDED FORMAT CONTROL OPERATIONS FOR CONSOLES

Terminal Class	Format Control Character	General Physical Operation	Code Substituted on Output Mechanism
1 and 2 7 and 8	LF	Space 1 line before next character output.	LF
	CR	Position to start of current line before next character output.	CR
4	LF	Space 1 line before next character output.	LF
	CR	Position to start of next line before next character output.	NL
5	LF	Space 1 line before next character output.	ESC, B
	CR	Position to start of current line before next character output.	ESC, G
6	LF	Space 1 line before next character output.	None
	CR	Position to start of current line before next character output.	CR
9 and 14	LF	Space 1 line before next character output.	None
	CR	Position to start of next line before next character output.	None
10 thru 13 and 15	LF	Space 1 line before next character output.	None
	CR	Position to start of next line line before next character output.	1B, 41 (ASCII); 31, 41 (External BCD)
16	LF	Space 1 line before next character output.	None
	CR	Position to start of next line before next character output.	10, 1F
17	LF	Space 1 line before next character output.	None
	CR	Position to start of next line before next character output.	10, 1E

Multiple-Message Input

For multiple-message input, the application program or the terminal user defines one or two input message-forwarding signals (equivalent to a normalized mode end-of-line indicator) and one or two transparent mode input delimiters. Each message ends at a message-forwarding signal; the last message ends when transparent input mode ends. The message-forwarding signal and mode delimiters are specified using the XL command options in section 3.

The possible message-forwarding signals are:

Occurrence of a specific character code in the input

Occurrence of a specific number of character bytes in the input

The transparent mode delimiters are:

Two consecutive occurrences of a specific character code (the message-forwarding signal)

A sequence of two character codes (a message-forwarding code followed by a transparent mode delimiter code)

Occurrence of a 200- to 400-millisecond timeout in the input

Upline Message Blocks

A transparent mode input block is assembled each time the network block size is reached or the Terminal Interface Program encounters a message-forwarding signal. The last block in the last message is assembled when the delimiter condition is encountered. If the message-forwarding signal is a specific character code, the TIP removes that code from the character stream before assembling the last block.

In transparent mode, the concept of a logical line is not meaningful to the network software. Both the end-of-line and end-of-block indicators are data within a transparent message.

Transparent Mode Output

Transparent mode output data can be divided arbitrarily into blocks and messages, provided the restrictions on network block size are met. A transparent mode downline block ends when the last character it contains is transferred to the network (defined by the tlc field in the block header, described later in this section).

If the TIP is performing page-wait operations for the terminal during transparent mode operation, output stops to wait for terminal operator acknowledgment at the end of each message. The automatic input feature can be used with the last block of a transparent mode output message.

Parity Processing

Actual terminal codes are right-justified with zero fill within the 8-bit character portion of the input or output byte. The codes contained in the input or output bytes depend on the parity option declared for the terminal.

The actual terminal code parity bit can be used for meaningful code only if no parity is declared (see the N option of the PA command in section 3). Otherwise, the parity bit is zero in input blocks and set by the Terminal Interface Program on output.

For example:

If the terminal uses a 7-bit code such as ASCII, with the eighth bit as a parity bit, the setting of the eighth bit is determined by the parity option selected for the terminal. If zero parity is declared, the eighth bit is always zero on input and output. If odd or even parity is declared, the eighth bit varies on input and output to satisfy the character parity requirement. If no parity is declared, the eighth bit is treated as part of the character data and is not changed during input or output.

If the terminal uses a 6-bit code, with the seventh bit as a parity bit, the setting of the seventh bit is determined by the parity option selected for the terminal. If zero parity is declared, the seventh bit is always zero on input and output. If odd or even parity is declared, the seventh bit varies on input and output to satisfy the character parity requirement. If no parity is declared, the seventh bit is treated as part of the character data and is not changed during input or output.

APPLICATION CHARACTER TYPES

Blocks always contain character bytes. These character bytes can be of several lengths and can be packed within bytes of several sizes. Each permitted combination of character byte length and packing byte size is called an application character type. There are several application character types supported by the released version of the software:

One 60-bit character byte per 60-bit word

One 8-bit character byte per 8-bit byte

One 8-bit character byte per 12-bit byte

One 6-bit display code character byte per 6-bit byte

Blocks transmitted through a network processing unit always consist of 8-bit characters in 8-bit bytes. An application program can use blocks of this application character type, or have NAM convert blocks to or from it so that the application program can use one of the remaining two application character types. Block conversion consists of byte mapping and character code conversion.

For a downline network data block, NAM:

Performs no mapping or character code conversion on 60-bit character bytes.

Performs no mapping or character code conversion on 8-bit characters in 8-bit bytes; the parity setting of the receiving device might cause the upper or eighth bit (bit 7) of the byte to be set.

Performs no character code conversion on 12-bit bytes but maps the 8-bit character to an 8-bit byte by discarding the leftmost four bits of the 12; the parity setting of the receiving device might cause the upper or eighth bit (bit 7) of the byte to be set.

Maps 6-bit characters to 8-bit characters by translating the former as 6-bit display code and substituting the corresponding hexadecimal code from the 128-character ASCII set.

For an upline network data block, NAM:

Performs no mapping or character code conversion on 60-bit character bytes.

Performs no mapping or character conversion on 8-bit characters in 8-bit bytes; the parity setting of the sending device might cause the upper or eighth bit (bit 7) of the byte to be set if the data is sent in transparent mode.

Performs character mapping but no code conversion by right-justifying 8-bit characters in 12-bit bytes with zero fill; the parity setting of the sending device might cause the upper or eighth bit (bit 7) of the byte to be set if the data is sent in transparent mode.

Maps and converts 8-bit characters to 6-bit characters by translating all ASCII control characters to display coded blanks, and translating all hexadecimal ASCII character codes between 60 and 7F to the display code equivalents of the hexadecimal ASCII character codes 40 to 5F. All other 7-bit ASCII codes are translated to the display codes equivalent to the CDC subset of the ASCII character set (refer to appendix A).

Because conversion and mapping between 6-bit and 8-bit characters involves a time-consuming character-by-character replacement of the block's data, use of a 6-bit display coded application character type is not recommended and is restricted to blocks exchanged with interactive devices. For efficiency, 8-bit byte characters are recommended for blocks exchanged with devices or other application programs through the interactive virtual terminal interface.

The application character type of an input block is determined by the character type associated with the logical connection. This association first occurs when the connection is established. You can change the association as necessary while the connection exists. The application character type of a specific input block is always indicated by a field in its associated block header word.

The application character type of an output block is determined solely by a field in its associated block header area. Input and output blocks transmitted over the same logical connection can therefore have different application character types.

CHARACTER BYTE CONTENT

Blocks containing 8-bit characters can be exchanged with an interactive device in normalized mode or in transparent mode. Blocks exchanged in normalized mode always contain 7-bit character codes from the ASCII character set, with the eighth bit set to zero. Blocks exchanged in transparent mode can contain 256 character codes from any character set used by a terminal, with the setting of the eighth bit determined by the parity processing selected for the device. Normalized mode exchanges are the initial mode for all logical connections. Blocks exchanged in transparent mode are identified by a field in their associated block header word.

The legal combinations of character types, modes, and uses are summarized in table 2-5. The mechanisms for declaring character types and exchange modes are described in the Block Header Content portion of this section and in section 3.

BLOCK HEADER CONTENT

The content of the block header word associated with a data block depends on whether the application program is sending or receiving the block. The requirements for all header words associated with upline data blocks are described in figure 2-3. The requirements for all header words associated with downline data blocks are described in figure 2-4.

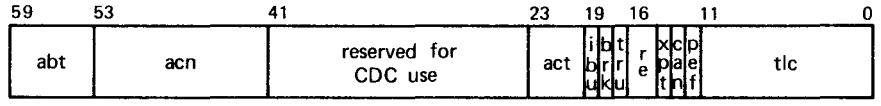
SUPERVISORY MESSAGE CONTENT AND SEQUENCE PROTOCOLS

Supervisory message blocks consist of 1 to 410 60-bit words or 1 to 2043 12-bit bytes. The fields within these blocks convey information and instructions to the network software, in a manner similar to the character bytes of a data message block. Supervisory messages are sent and received through the same application program routines as are used for data blocks. (See sections 4 and 5.) Supervisory messages have associated block header words, just as data blocks do. These header words convey information to the network software concerning the contents of the corresponding text area buffer.

Supervisory messages have the general formats shown in figures 2-5 and 2-6. A specific message contains a fixed combination of four fields and can include additional parameters. The individual messages supported by the network software are described in section 3. The fields are described below in the order of their use, rather than in the order of their occurrence within a supervisory message.

TABLE 2-5. CHARACTER EXCHANGES WITH CONNECTIONS

Application Character Type	ACT Field Value	Exchange Mode Used	Connection Type	Code Set (Character Set)
60-bit characters in 60-bit bytes	1	Normalized	Application-to-application within the same host	Binary (None)
8-bit characters in 8-bit byte	2	Normalized	Application-to-terminal (consoles)	7-bit ASCII (128 ASCII)
8-bit characters in 8-bit bytes	2	Transparent	Application-to-terminal (consoles)	Any 6-, 7-, or 8-bit (Unknown)
8-bit characters in 8-bit bytes	2	Normalized	Application-to-application	Binary (None)
8-bit characters in 12-bit bytes	3	Normalized	Application-to-terminal (consoles)	7-bit ASCII (128 ASCII)
8-bit characters in 12-bit bytes	3	Transparent	Application-to-terminal (consoles)	Any 6-, 7-, or 8-bit (Unknown)
8-bit characters in 12-bit bytes	3	Normalized	Application-to-application	Binary (None)
6-bit characters in 6-bit bytes	4	Normalized	Application-to-terminal (consoles)	6-bit display code to/from 7-bit ASCII (64-character subset of ASCII)



ha — Symbolic header area address, specified as the location to receive the application block header in a call to NETGET, NETGETL, NETGETF, or NETGTFL (see section 5).

abt — Application block type of the associated network data block. This field can have the values:

- =0 indicates a null block. (No block is queued or none can be delivered from the logical connection polled.)
- =1 indicates that the associated block is one of several blocks comprising a single message, but is not the last such block.
- =2 indicates that the associated block is either the last or only one comprising the message.

Values of 3 through 63 are not valid for data blocks on input. You can access this field with the reserved symbol ABHABT (see section 4).

acn — Application connection number of the logical connection from which the associated block was sent. This field can have the values $1 \leq minacn \leq acn \leq maxacn \leq 4095$, where the values minacn and maxacn are parameters in the NETON statement (see section 5). You can access this field with the reserved symbol ABHADR (see section 4).

Figure 2-3. Application Block Header Content for Upline Network Data Blocks (Sheet 1 of 4)

act — Application character type used to encode the accompanying block. This field can contain the values:

- =1 — 60-bit transparent characters, packed one per central memory word; this character type can be used only for application-to-application connections within the same host.
- =2 — 8-bit characters, packed 7.5 per central memory word; this character type is recommended for terminal-to-application connections.
- =3 — 8-bit characters, right-justified in 12-bit bytes with zero fill, packed 5 per central memory word.
- =4 — 6-bit display code characters (see table A-1 in appendix A), packed 10 per central memory word. This value can be used only for terminal-to-application connections in normalized mode when the block is exchanged with a site-defined device or a CDC-defined console device.
- =5 thru 11 — Reserved for CDC use; not currently recognized.
- =12 thru 15 — Reserved for installation use; usage and content are unrestricted and undefined (the released version of the software does not recognize these values).

The value contained in the act field is the value assigned to the connection by the application program for input, either in the connection-accepted supervisory message (ict field) or in the most recent change-input-character-type supervisory message (see section 3). You can access this field with the reserved symbol ABHACT (see section 4).

ibu — Input-block-undeliverable bit. When ibu has a value of 1, the block associated with this block header has not been delivered to the application program; ibu is 1 when the block:

- Is larger than the maximum text length (tlmax parameter) declared by the application program in its NETGET, NETGETL, NETGETF, or NETGTFL call and the program has not requested that input data be truncated (see the truncate-input asynchronous supervisory message described in section 3). The block header contains the actual length of the queued block in its tlc field, given in character units specified by the act field. The block remains queued until the application program takes one of the following actions:

  Uses the change-input-character-type asynchronous supervisory message described in section 3 to compress the characters into fewer central memory words by using a different application character type to pack them more densely.

  Uses the input-truncation asynchronous supervisory message described in section 3 to delete enough characters so that the remainder fit into the existing text area.

  Uses a longer text area.

  The application program then must use another NETGET, NETGETL, NETGETF, or NETGTFL call to obtain the block.

- Contains transparent mode data from a connection using an act value of 4. The block header contains the actual length of the queued block in its tlc field (given in 8-bit bytes) and has an xpt value of 1 (see xpt field description). The application program can:

  Change the input character type for the connection to a value of 2 or 3, using the change-input-character-type asynchronous supervisory message described in section 3, then use a NETGET, NETGETL, NETGETF, or NETGTFL call to obtain the block.

  Use the change-input-character-type asynchronous supervisory message with a set nxp bit as described in section 3; this discards the queued block and all subsequent blocks of transparent data from the connection.

Figure 2-3. Application Block Header Content for Upline Network Data Blocks (Sheet 2 of 4)

- Is queued on a connection between application programs within the same host and the act value specified by your application does not match the act value specified by the other application in its NETPUT call for the block. The application program can:

  Change the input character type for the connection using the change-input-character-type asynchronous supervisory message described in section 3, then use a NETGET, NETGETL, NETGETF, or NETGTFL call to obtain the block.

You can access this field with the reserved symbol ABHIBU (see section 4).

brk — Break occurred bit. When brk is 1, the application program receives an asynchronous break supervisory message concerning this connection (see section 3); that message contains a reason code explaining the reason for the break . A brk value of 1 only occurs in the header for a block with an abt value of 0. This value indicates that the associated null block was sent upline to mark when the break condition occurred on the connection. You can access the brk field with the reserved symbol ABHBRK (see section 4).

tru — Truncated data bit. When tru is 1, the block associated with this block header has been truncated to fit into the text area used. When tru is 0, the block has not been truncated. The tru bit cannot be 1 unless the application program has issued the data truncation control asynchronous supervisory message described in section 3 and that message affects transmissions on this connection. When truncation occurs, the tlc field contains the maximum number of complete transferred character bytes of the block. You can access the tru field with the reserved symbol ABHTRU (see section 4).

re — Reserved for CDC use.

xpt — Transparent mode bit, indicating whether the accompanying block contains transparent mode data. If your program chooses not to receive transparent mode input when it accepts a connection or changes the input character type of the connection (nxp field, described in section 3), an xpt value of 1 is received in a block with an abt of 0 (an empty block) and indicates that one or more transparent mode blocks were discarded by the network software.

If your program can receive transparent mode input, the interpretation of the value this field contains depends on the act value used, as follows:

act=1, xpt should be ignored.

act=2, if the data is from a site-defined device or a CDC-defined console device:

xpt=0 indicates normalized mode data for which interactive virtual terminal transformations were performed; 7-bit characters are from the 128-character ASCII set (see appendix A).

xpt=1 indicates transparent mode data for which no transformations were performed; all eight bit positions might be used to form 256 characters, but the application program must correctly interpret the format of such data.

act=2, if the data is from an application program:

xpt=0 indicates that the sending application program did not use an xpt value of 1 in its block header for the accompanying block.

xpt=1 indicates that the sending application program used an xpt value of 1 in its block header for the accompanying block.

act=3, if the data is from a site-defined device or a CDC-defined console device:

xpt=0 indicates normalized mode data for which interactive virtual terminal transformations were performed; 7-bit characters are from the 128-character ASCII set (see appendix A).

Figure 2-3. Application Block Header Content for Upline Network Data Blocks (Sheet 3 of 4)

xpt=1 indicates transparent mode data for which no transformations were performed; all eight bit positions in the character portion of the character byte might be used to form 256 characters, but the application program ,ust correctly interpret the format of such data.

act=3, if the data is from an application program:

xpt=0 indicates that the sending application program did not use an xpt value of 1 in its block header for the accompanying block.

xpt=1 indicates that the sending application program used an xpt value of 1 in its block header for the accompanying block.

act=4, if the data is from a site-defined device or a CDC-defined console device:

xpt=0 indicates normalized mode data for which interactive virtual terminal transformations were performed; 6-bit characters are from the 6-bit display code set (see table A-1 in appendix A).

xpt=1 indicates that the ibu bit is also set; the tlc field contains the actual block length in 8-bit characters (not in 6-bit characters). Transparent mode is not supported for act=4; a change-input-character-type supervisory message must be issued before the block can be received (see section 3).

You can access this field with the reserved symbol ABHXPT (see section 4).

can Cancel-input bit. When can is 1, the terminal operator used the cancel-input key defined for the device or the break condition key (see BR command in section 3) to end the text in the associated block. The associated block always has an abt of 2, and the data is always from a console device. The cancel-input request also applies to any blocks with an abt value of 1 that preceded this block; all blocks in the same message should be discarded. You can access this field with the reserved symbol ABHCAN (see section 4).

pef Parity error flag bit. When pef is 1, the associated block contains a parity error in one or more of its characters. You can access this field with the reserved symbol ABHBIT (see section 4).

tlc Text length of the associated block, in character units specified by the act field. The equivalent length in central memory words can be computed as follows:

act=1, tlc is the number of central memory words the block requires.

act=2, the number of central memory words the block requires is tlc divided by 7.5, rounded upward to an integer.

act=3, the number of central memory words the block requires is tlc divided by 5, rounded upward to an integer.

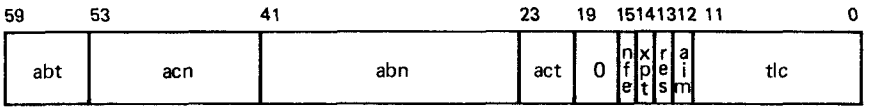
act=4, the number of central memory words the block requires is tlc divided by 10, rounded upward to an integer.

=5 thru 11 Reserved for CDC use; not currently recognized.

=12 thru 15 Reserved for installation use; usage and content are undefined.

You can access this field with the reserved symbol ABHTLC (see section 4).

Figure 2-3. Application Block Header Content for Upline Network Data Blocks (Sheet 4 of 4)



ha — Symbolic header area address, specified as the application block header's location in a call to NETPUT or NETPUTF (see section 5).

abt — Application block type of the accompanying network data block. This field can contain the values:

- =1, indicates that the accompanying block is one of several blocks comprising a single message, but is not the last such block.
- =2, indicates that the accompanying block is either the last or only one comprising a message.

Values of 0 and 3 through 63 are not valid for data blocks on output. You can access this field with the reserved symbol ABHABT (see section 4).

acn — Application connection number of the logical connection to which the accompanying block should be sent. This field can contain the values $1 \leq minacn \leq acn \leq maxacn \leq 4095$, where the values minacn and maxacn are parameters in the NETON statement (see section 5.) You can access this field with the reserved symbol ABHADR (see section 4).

abn — Application block number assigned to the block being sent. This field is an 18-bit integer that identifies the block when the network software's processing of the block returns certain supervisory messages (see section 3). You define the block number; it can be:

- A sequencing number
- The block's central memory address
- The block's mass storage address (physical record unit)
- An index value for a block control array or table
- An external label

You can access this field with the reserved symbol ABHABN (see section 4).

act — Application character type used to encode the accompanying block. This field can contain the values:

- =1, 60-bit transparent characters, packed one per central memory word; this character type can be used only for application-to-application connections within the same host.
- =2, 8-bit characters, packed 7.5 per central memory word; this character type is recommended for application-to-terminal connections.
- =3, 8-bit characters, right-justified in 12-bit bytes, packed 5 per central memory word.
- =4, 6-bit display code characters (see table A-1 in appendix A), packed 10 per central memory word. This value can be used only for normalized mode data on application-to-terminal connections when the block is exchanged with a site-defined device or a CDC-defined console device.
- =5 thru 11 Reserved for CDC use; not currently recognized.
- =12 thru 15 Reserved for installation use; usage and content are unrestricted and undefined (the released version of the software does not recognize these values).

You can access this field with the reserved symbol ABHACT (see section 4).

Figure 2-4. Application Block Header Content for Downline Network Data Blocks (Sheet 1 of 3)

nfe — No-format-effector bit, indicating whether the accompanying block contains format effectors. If nfe is 1, there are no format effectors in the block; if nfe is 0, the block contains format effectors requiring removal and interpretation. The nfe field applies only to normalized mode data exchanged with a site-defined device or a CDC-defined console device. You can access this field with the reserved symbol ABHNFE (see section 4).

xpt — Transparent mode bit, indicating whether the accompanying block contains transparent mode data. The value used in this field depends on the act value used, as follows:

- act=1, xpt value is ignored and can be 1 or 0.
- act=2, if the data is for a site-defined device or a CDC-defined console device:
  - xpt=0 indicates normalized mode data for which interactive virtual terminal transformations should be performed; 7-bit characters are from the 128-character ASCII set (see appendix A).
  - xpt=1 indicates transparent mode data for which no transformations are to be performed; all eight bit positions can be used to form 256 characters, but such data must be correctly formatted for terminal output.
- act=2, if the data is for an application program, xpt should be 1 only if the receiving program can detect and support that value.
- act=3, if the data is for a site-defined device or a CDC-defined console device:
  - xpt=0 indicates normalized mode data for which interactive virtual terminal transformations should be performed; 7-bit characters are from the 128-character ASCII set (see appendix A).
  - xpt=1 indicates transparent mode data for which no transformations are performed; all eight bit positions in the character portion of the character byte can be used to form 256 characters, but such data must be correctly formatted for terminal output.
- act=3, if the data is for an application program, xpt should be 1 only if the receiving program can detect and support that value.
- act=4, xpt value is ignored and can be 1 or 0.

You can access this field with the reserved symbol ABHXPT (see section 4).

res — Reserved for CDC use.

aim — Automatic-input-mode flag bit. You can use this field when the accompanying block is the last block (abt of 2) of a message sent to a site-defined device or a CDC-defined console device and contains only one logical line. If aim is 1, the first text characters (excluding format effectors) of the block become the first characters of the next data block input from the device. If the block contains fewer than 20 characters, only the characters present are used; if the block contains more than 20 characters, only the first 20 are used. When the downline block contains transparent mode data, the next input block will not be in transparent mode unless transparent mode input operation has been explicitly selected by the terminal operator or the application program (with one of the supervisory messages described in section 3). The aim value is ignored for blocks with an abt of 1. You can access this field with the reserved symbol ABHBIT (see section 4).

Figure 2-4. Application Block Header Content for Downline Network Data Blocks (Sheet 2 of 3)

tlc — Text length of the associated block, in character units specified by the act value. The value to use in the tlc field can be computed as follows:

act=1, tlc is the number of central memory words occupied by the block.

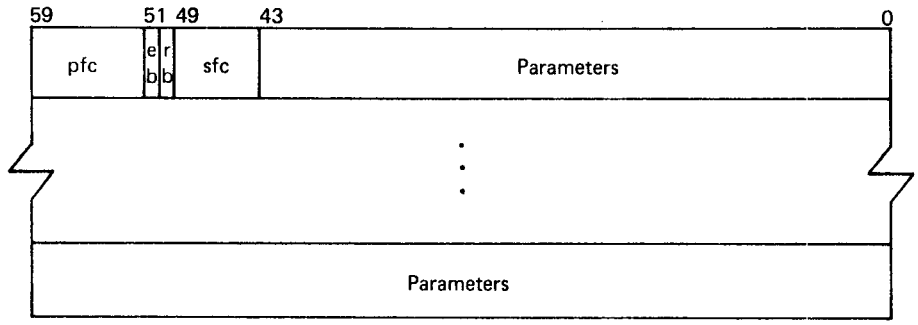
act=2, tlc is the number of complete central memory words occupied by the block times 7.5, plus the number of complete character bytes used in any remaining central memory word, rounded upward to an integer.

act=3, tlc is the number of complete central memory words occupied by the block times 5, plus the number of 12-bit character bytes used in any remaining central memory word.

act=4, tlc is the number of complete central memory words occupied by the block times 10.

The character count used as the text length must include any format effectors and end-of-line indicator bytes contained in the block. You can access this field with the reserved symbol ABHTLC (see section 4).

Figure 2-4. Application Block Header Content for Downline Network Data Blocks (Sheet 3 of 3)



ta — Symbolic text area address, specified in a NETGET, NETGETF, NETGETL, or NETGTFL call as the location to receive an upline supervisory message or specified in a NETPUT or NETPUTF call as the location from which to send a downline supervisory message (see section 5).

pfc — Primary function code. Field mnemonics are used throughout this manual in specific message formats. Reserved symbols corresponding to the field mnemonics can be used to access message fields (see section 4). Reserved symbols for the primary function code are used throughout this manual within mnemonics identifying specific messages. The mnemonics and their unpacked (right-justified) numerical equivalents are:

Field Mnemonic	Reserved Symbolic Mnemonic	Octal	Hexadecimal	Decimal
con	CON	143	63	099
ctrl†	CTRL	301	C1	193
dc	DC	302	C2	194
err	ERR	204	84	132
fc	FC	203	83	131
hop	HOP	320	D0	208
intr	INTR	200	80	128
lst	LST	300	C0	192
shut	SHUT	102	42	066
tch	TCH	144	64	100
to†	TO	304	C4	196

Primary function codes 00 through E0 hexadecimal are reserved for CDC use. Hexadecimal codes E1 through EF are for installation use and have no predefined meanings or reserved symbols. You can access the pfc field with the reserved symbol PFC (see section 4).

Figure 2-5. Supervisory Message General Content, Asynchronous Messages and Synchronous Messages of Application Character Type 2 (Sheet 1 of 2)

eb — Error bit. When set to 1, eb indicates the occurrence of an error (an abnormal response to a previous supervisory message); when set to 0, eb indicates a normal response. The eb field always contains 0 when a supervisory message is not a response to a prior message. You can access this field with the reserved symbol EB (see section 4).

rb — Response bit. When set to 1, rb indicates a normal response to a previous supervisory message; rb is always 0 in a supervisory message that is not a response to a previous message. You can access this field with the reserved symbol RB (see section 4).

sfc — Secondary function code. Field mnemonics are used throughout this manual in specific message formats. Reserved symbols corresponding to the field mnemonics can be used to access message fields (see section 4). Reserved symbols for the secondary function code are used throughout this manual within mnemonics identifying specific messages. The sfc mmemonics and their unpacked (right-justified) numerical equivalents are:

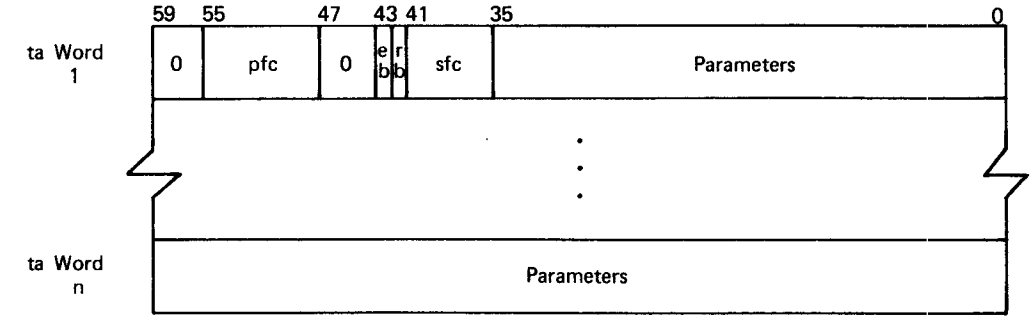
Field Mnemonic	Related Symbolic pfc	Reserved Symbolic Mnemonic	Octal	Hexadecimal	Decimal
req	CON	REQ	00	00	00
acrq	CON	ACRQ	02	02	02
cb	CON	CB	05	05	05
end	CON	END	06	06	06
def†	CTRL	DEF	04	04	04
char†	CTRL	CHAR	10	08	08
rtc†	CTRL	RTC	11	09	09
tcd†	CTRL	TCD	12	0A	10
cict	DC	CICT	00	00	00
tru	DC	TRU	01	01	01
lgl	ERR	LGL	01	01	01
brk	FC	BRK	00	00	00
rst	FC	RST	01	01	01
ack	FC	ACK	02	02	02
nak	FC	NAK	03	03	03
inact	FC	INACT	04	04	04
init	FC	INIT	07	07	07
db	HOP	DB	16	0E	14
de	HOP	DE	17	0F	15
du	HOP	DU	03	03	03
trace	HOP	TRACE	02	02	02
notr	HOP	NOTR	07	07	07
rel	HOP	REL	15	0D	13
rs	HOP	RS	10	08	08
usr	INTR	USR	00	00	00
rsp	INTR	RSP	01	01	01
app	INTR	APP	02	02	02
off	LST	OFF	00	00	00
on	LST	ON	01	01	01
swh	LST	SWH	02	02	02
fdx	LST	FDX	03	03	03
hdx	LST	HDX	04	04	04
insd	SHUT	INSD	06	06	06
tchar	TCH	TCHAR	00	00	00
mark†	TO	MARK	00	00	00

You can access the sfc field with the reserved symbol SFC (see section 4).

parameters — These parameters can extend into words 2 through n; $n \leq 410$. Parameters are defined in the descriptions of the specific messages in section 3.

†Synchronous supervisory message fields.

Figure 2-5. Supervisory Message General Content, Asynchronous Messages and Synchronous Messages of Application Character Type 2 (Sheet 2 of 2)



ta — Symbolic text area address, specified in a NETGET, NETGETF, NETGETL, or NETGTFL call as the location to receive an upline supervisory message or specified in a NETPUT or NETPUTF call as the location from which to send a downline supervisory message (see section 5).

pfc — Primary function code. Field mnemonics are used throughout this manual in specific message formats. Reserved symbols corresponding to the field mnemonics can be used to access message fields (see section 4). Reserved symbols for the primary function code are used throughout this manual within mnemonics identifying specific messages. The mnemonics and their unpacked (right-justified) numerical equivalents are:

Field Mnemonic	Reserved Symbolic Mmemonic	Octal	Hexadecimal	Decimal
ctrl	CTRL	301	C1	193
to	TO	304	C4	196

Primary function codes 00 through E0 hexadecimal are reserved for CDC use. Hexadecimal codes E1 through EF are for installation use and have no predefined meanings or reserved symbols. You can access the pfc field with the reserved symbol PFC (see section 4).

eb — Error bit. When set to 1, eb indicates the occurrence of an error (an abnormal response to a previous supervisory message); when set to 0, eb indicates a normal response. The eb field always contains 0 when a supervisory message is not a response to a prior message. You can access this field with the reserved symbol EB (see section 4).

rb — Response bit. When set to 1, rb indicates a normal response to a previous supervisory message; rb is always 0 in a supervisory message that is not a response to a previous message. You can access this field with the reserved symbol RB (see section 4).

sfc — Secondary function code. Field mnemonics are used throughout this manual in specific message formats. Reserved symbols corresponding to the field mnemonics can be used to access message fields (see section 4). Reserved symbols for the secondary function code are used throughout this manual within mnemonics identifying specific messages. The sfc mmemonics and their unpacked (right-justified) numerical equivalents are:

Field Mnemonic	Related Symbolic pfc	Reserved Symbolic Mnemonic	Octal	Hexadecimal	Decimal
def	CTRL	DEF	04	04	04
char	CTRL	CHAR	10	08	08
rtc	CTRL	RTC	11	09	09
tcd	CTRL	TCD	12	0A	10
mark	TO	MARK	00	00	00

You can access the sfc field with the reserved symbol SFC (see section 4).

parameters — These parameters can extend into words 2 through n; $n \leq 410$. Parameters are defined in the descriptions of the specific messages in section 3.

Figure 2-6. Supervisory Message General Content, Synchronous Messages of Application Character Type 3

The first of the four fields common to all supervisory messages is the primary function code. The primary function code is used to group supervisory messages into related functions and determine their routing within the network software.

Functions routed between NAM and the application program are represented in figures 2-5 and 2-6 by mnemonics. These mnemonics are defined in parentheses after the corresponding function in the following list:

Connection data flow control (FC)

Error reporting (ERR)

Device control (CTRL)

Connection list management (LST)

Connection characteristic definition (DC)

Interrupt request (INTR)

Connection control (CON)

Terminal characteristic definition (TCH)

Network shutdown (SHUT)

Host operator commands (HOP)

Terminate output (TO)

The precise function of a message within a primary function grouping is indicated by its secondary function code, forming the fourth common field. The mnemonic symbols used to identify these secondary function codes are related to the use of the messages. Mnemonics for these codes also appear in figures 2-5 and 2-6 and in parentheses after the secondary functions in the following list:

Request for logical connection (REQ)

End of connection (END)

Connection broken (CB)

Application-to-application connection request (ACRQ)

Internal shutdown (INSD)

Inactive connection (INACT)

No acknowledgment (NAK)

Acknowledgment (ACK)

Reset (RST)

Break (BRK)

Logical problem (LGL)

Initialization (INIT)

Mark point in data (MARK)

Switch connection between lists (SWH)

Turn connection list processing off (OFF)

Turn connection list processing on (ON)

Turn half-duplex operation on for connection on a list (HDX)

Turn full-duplex operation on for connection on a list (FDX)

Begin truncating input on a connection (TRU)

Application interrupt request (APP)

User interrupt request (USR)

Interrupt response (RSP)

Change input character type (CICT)

Report of changed terminal characteristics (TCHAR)

Request terminal characteristics (RTC)

Define single terminal characteristic (DEF)

Upline terminal multiple characteristics definition (TCD)

Downline terminal multiple characteristics definition (CHAR)

The second and third common fields are used to indicate whether the function was performed or not. By convention, these fields are called the error and response bits. The error bit is usually set to indicate the message recipient's refusal to perform the function; the response bit is set to indicate the recipient's normal completion of the function.

Together, the four common fields define one supervisory message. Supervisory messages can be grouped into two classes of sequencing protocol:

Asynchronous (the largest class)

Synchronous

ASYNCHRONOUS MESSAGES

Asynchronous supervisory messages are sent or received separately from the stream of data message blocks between an application program and a logical connection. Whether these messages are used alone or as part of the stereotyped sequences described in section 3, their receipt or the need to send them cannot be predicted from the generalized logic required for data block processing. Such messages are said to be asynchronous to the data block stream.

All asynchronous messages are sent or received on a special logical connection with the preassigned application connection number of zero. The network software preassigns this application connection number to connection list zero.

All asynchronous supervisory messages are actually sent to or received from software resident in the host computer, although they may be reformatted by this software for communication with software outside of the host. These messages conform to the requirements of application-to-application connections. Asynchronous supervisory messages therefore use an application character type of one. All supervisory messages are assigned the nonzero application block type of three.

Asynchronous supervisory messages are processed with the same AIP routines used by an application program to process data message blocks on logical connections other than application connection number zero. Asynchronous supervisory messages are queued on their special connection until fetched by the application program.

The application program fetches supervisory messages one message at a time. When the connection queue is empty, a null block with an application block type of zero is returned.

The network software provides a mechanism for the application program to determine when asynchronous supervisory messages are queued on application connection number zero. When a call to an AIP routine is completed, a supervisory status word at a location defined by the application program is updated to indicate whether any asynchronous supervisory messages are queued. As long as the application program continues to make calls to AIP routines, it can test the supervisory status word periodically (instead of attempting to fetch null blocks from application connection number zero). The supervisory status word and the use of NETWAIT are described in section 5.

SYNCHRONOUS MESSAGES

Synchronous supervisory messages are sent or received embedded in the stream of data message blocks between an application program and a logical connection. Whether these messages are used alone or as part of the stereotyped sequences described in section 3, their receipt or the need to send them is determined by the generalized logic required for data block processing. Such messages are said to be synchronous with the data block stream.

All synchronous messages are sent or received on the logical connection to which they apply. This logical connection cannot be application connection number zero.

All synchronous supervisory messages are actually sent to or received from network software outside of the host computer. Because the application program processes these messages as network blocks sent to or received from terminals, the messages conform to the requirements of application-to-terminal connections. Synchronous supervisory messages use an application character type of two or three; your program specifies which is used when it accepts the connection to the terminal.

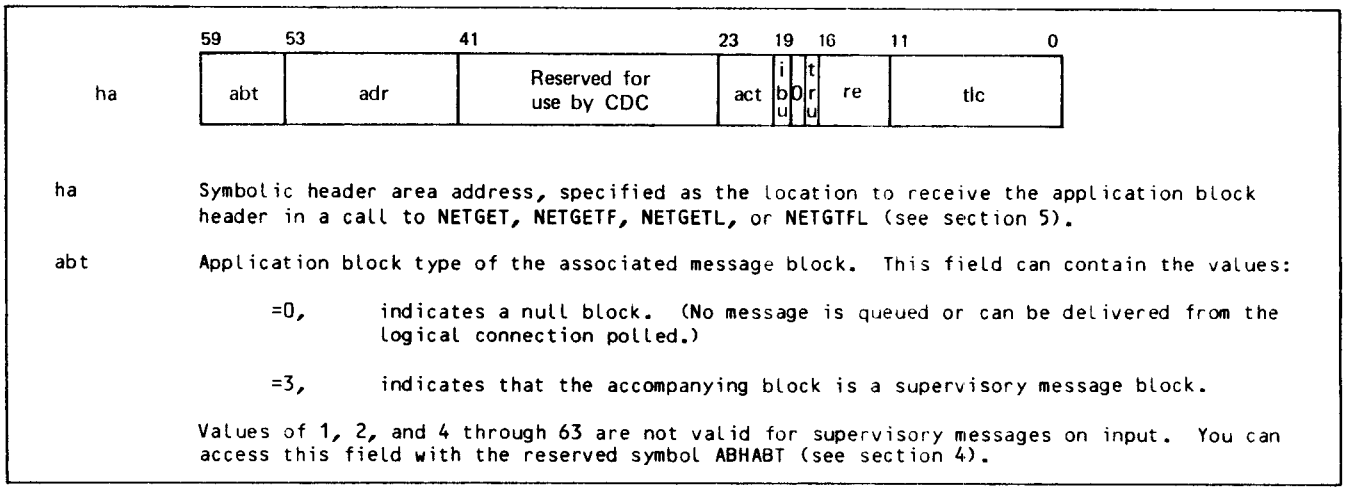
Synchronous supervisory messages are processed with the same AIP routines used by an application program to process other blocks on logical connections. Synchronous supervisory messages are queued on their connections until fetched by the application program. Because the application program must distinguish between data or null blocks and synchronous supervisory message blocks, supervisory messages are assigned the application block type of three.

The network software provides a mechanism for the application program to determine when synchronous supervisory messages or data blocks are queued on a logical connection. When a call to the AIP routine NETWAIT is completed, a supervisory status word at a location defined by the application program is updated to indicate whether any synchronous supervisory message or data blocks are queued. The application program can test the supervisory status word periodically, instead of attempting to fetch null blocks from all application connection numbers. The supervisory status word and the use of NETWAIT are described in section 5.

Synchronous supervisory messages are subject to the same application block limit as data messages and are similarly acknowledged. This process is described in section 3.

BLOCK HEADER CONTENT

The content of the block header word associated with a supervisory message depends on whether the message is asynchronous or synchronous, and on whether it is being sent or received. The requirements for asynchronous and synchronous messages are described in the preceding subsection. The requirements for all header words associated with incoming supervisory messages are described in figure 2-7. The requirements for all header words associated with outgoing supervisory messages are described in figure 2-8.



ha — Symbolic header area address, specified as the location to receive the application block header in a call to NETGET, NETGETF, NETGETL, or NETGTFL (see section 5).

abt — Application block type of the associated message block. This field can contain the values:

- =0, indicates a null block. (No message is queued or can be delivered from the logical connection polled.)
- =3, indicates that the accompanying block is a supervisory message block.

Values of 1, 2, and 4 through 63 are not valid for supervisory messages on input. You can access this field with the reserved symbol ABHABT (see section 4).

Figure 2-7. Application Block Header Content for Upline Supervisory Messages (Sheet 1 of 2)

adr — Application connection number of the logical connection from which the message block comes. This field can have the values:

- =0, for asynchronous supervisory messages from the host portion of the network software.
- =acn, for synchronous supervisory messages from the Terminal Interface Program servicing the logical connection with the indicated nonzero application connection number.

You can access this field with the reserved symbol **ABHADR** (see section 4).

act — Application character type used to encode the accompanying message block. The value appearing in this field depends on the type of supervisory message involved and on the act value you chose (the sct field described in section 3) for synchronous supervisory messages on this connection; this field can contain the values:

- =1, an asynchronous supervisory message packed in 60-bit words. Must be used for supervisory messages with an adr value of 0.
- =2, a synchronous supervisory message packed in 8-bit characters, 7.5 characters per central memory word (the recommended value).
- =3, a synchronous supervisory message packed in 8-bit characters, 5 characters per central memory word.

Because the fields within supervisory messages are groups of bits within central memory words (rather than characters in a character string), the act field of a supervisory message does not indicate that character mapping occurred. You can access this field with the reserved symbol **ABHACT** (see section 4).

ibu — Input-block-undeliverable bit. When ibu is 1, the block associated with this block header has not been delivered to the application program. The block is larger than the maximum text length (tlmax parameter) declared by the application program in its **NETGET, NETGETF, NETGETL,** or **NETGTFL** call and remains queued until:

A **NETGET, NETGETL, NETGETF,** or **NETGTFL** call occurs for the connection and specifies an adequate text length (see section 5).

A truncate-input asynchronous supervisory message (see section 3) is issued for the connection and a **NETGET, NETGETL, NETGETF,** or **NETGTFL** call occurs for the connection (see section 5). This action resolves the problem only for synchronous supervisory messages.

A block header with an ibu value of 1 contains the actual length of the queued block in its tlc field, given in character units specified by the act field. You can access this field with the reserved symbol **ABHIBU** (see section 4).

tru — Truncated data bit. When tru is 1, the synchronous supervisory message block associated with this block header has been truncated to fit into the text area used. Asynchronous supervisory messages are never truncated. This bit contains a meaningful value only after the application program has issued the data truncation control asynchronous supervisory message described in section 3 and only if that message affects transmissions on this connection. When truncation occurs, the block header for the truncated block contains the maximum number of complete transferred character bytes in its tlc field. You can access this field with the reserved symbol **ABHTRU** (see section 4).

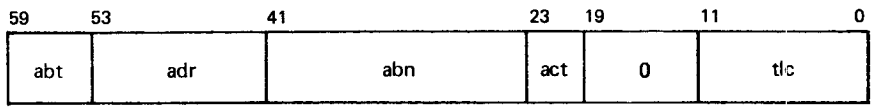
re — Reserved for CDC use.

tlc — Text length of the associated block, in character units specified by the act field, as follows:

- act=1, tlc is the number of central memory words occupied by the block.
- act=2, tlc is the number of 8-bit bytes containing meaningful message fields.
- act=3, tlc is the number of 12-bit bytes containing meaningful message fields.

You can access this field with the reserved symbol **ABHTLC** (see section 4).

Figure 2-7. Application Block Header Content for Upline Supervisory Messages (Sheet 2 of 2)



ha — Symbolic header area address, specified as the application block header's location in a call to NETPUT or NETPUTF (see section 5).

abt — Application block type; abt is 3 for all supervisory messages. You can access this field with the reserved symbol ABHABT (see section 4).

adr — Application connection number of the logical connection to which the message block should be sent. This field can contain the values:

- =0, for asynchronous supervisory messages addressed to the host portion of the network software.
- =acn, for synchronous supervisory messages addressed to the Terminal Interface Program servicing the logical connection with the indicated nonzero application connection number.

You can access this field with the reserved symbol ABHADR (see section 4).

abn — Application block number assigned to the message block being sent. This field is an 18-bit integer that identifies a synchronous supervisory message block when the network software's processing of the block returns a block-delivered or block-not-delivered supervisory message. This field is generally ignored for asynchronous supervisory messages. If the message is a request for connection with another application program, that application program will receive this integer as part of the request; see the CON/ACRQ/R supervisory message description in section 3. You define the block number; it can be:

- A sequencing number
- The block's central memory address
- The block's mass storage address (physical record unit)
- An index value for a block control array or table
- An external label

You can access this field with the reserved symbol ABHABN (see section 4).

act — Application character type used to encode the accompanying message block. The value declared for this field depends on the type of supervisory message involved; this field can have the values:

- =1, an asynchronous supervisory message packed in 60-bit transparent character bytes, one character per central memory word.
- =2, a synchronous supervisory message packed in 8-bit character bytes, 7.5 bytes per central memory word; the recommended value.
- =3, a synchronous supervisory message packed in 8-bit characters within 12-bit bytes, 5 bytes per central memory word.

You can access this field with the reserved symbol ABHACT (see section 4).

tlc — Text length of the accompanying block, in character units specified by the act field, as follows:

- act=1, tlc is the number of central memory words occupied by the block.
- act=2, tlc is the number of 8-bit bytes containing meaningful message fields.
- act=3, tlc is the number of 12-bit bytes containing meaningful message fields.

You can access this field with the reserved symbol ABHTLC (see section 4).

Figure 2-8. Application Block Header Content for Downline Supervisory Messages

This section describes all synchronous and asynchronous supervisory messages that are legal for application program communication with network software. These messages are described in the context of their use.

MESSAGE PROTOCOLS

Figure 2-5 in section 2 shows the general format of a supervisory message. Note that this information is in the text area of the message and must be accompanied by an application block header as described in section 2. A supervisory message is identified by the contents of its primary function code field, error bit, response bit, and secondary function code field. This allows a supervisory message to be described by a mnemonic of the form shown in figure 3-1. Although many combinations of valid field values are possible, only certain combinations are permitted. Table 3-1 lists these legal messages alphabetically by mnemonic.

MESSAGE SEQUENCES

Supervisory messages are always used in stereotyped sequences of one or more messages. Related messages (messages distinguished by the use of the error or response bits) are always part of multiple-message sequences. The messages described in the following subsections are discussed in the context of their normal sequences. Each sequence is illustrated with a figure that shows the sender and recipient of the messages in the sequence, and the direction of transmission of each message (arrows).

Message sequences include the following:

- Managing logical connections
- Managing connection lists
- Controlling data flow
- Converting data
- Truncating data
- Changing terminal characteristics
- Requesting terminal characteristics
- Host operator communication
- Host shutdown
- Error reporting

pfc/sfc/sm

pfc The reserved symbolic mnemonic for the contents of the primary function code field; this mnemonic can be any of those listed in figure 2-5 in section 2.

sfc The reserved symbolic mnemonic of the contents of the secondary function code field; this mnemonic can be any of those listed in figure 2-5 in section 2, provided the secondary function code is legal for the primary function code used.

sm A letter indicating the combined settings of the error and response bits; this letter can be:

- R Indicating an initial request supervisory message (bit setting 00)
- N Indicating a normal response supervisory message (bit setting 01)
- A Indicating an abnormal response supervisory message (bit setting 10)

Figure 3-1. Supervisory Message Mnemonic Structure

TABLE 3-1. LEGAL SUPERVISORY MESSAGES

Message Mnemonic	Message Meaning	Type	Octal Equivalent of Bits 59 thru 42 of First Text Area Word†	Block Header Fields	Figure Number Defining Message
CON/ACRQ/A	Rejection of application-to-application connection request	Upline asynchronous	307010	acn = 0 act = 1 tlc = 2	3-13
CON/ACRQ/R	Application-to-application connection request	Downline asynchronous	306010	acn = 0 act = 1 tlc = 2	3-12

TABLE 3-1. LEGAL SUPERVISORY MESSAGES (Contd)

Message Mnemonic	Message Meaning	Type	Octal Equivalent of Bits 59 thru 42 of First Text Area Word†	Block Header Fields	Figure Number Defining Message
CON/CB/R	Connection broken	Upline asynchronous	306024	acn = 0 act = 1 tlc = 1	3-8
CON/END/N	All connection processing completed	Upline asynchronous	306430	acn = 0 act = 1 tlc = 1	3-10
CON/END/R	End all connection processing	Downline asynchronous	306030	acn = 0 act = 1 tlc $\geq$ 2	3-9
CON/REQ/A	Connection rejected	Downline asynchronous	307000	acn = 0 act = 1 tlc = 1	3-5
CON/REQ/N	Connection accepted	Downline asynchronous	306400	acn = 0 act = 1 tlc = 1	3-4
CON/REQ/R	Connection requested	Upline asynchronous	306000	acn = 0 act = 1 tlc $\geq$ 6	3-3
CTRL/CHAR/A	No terminal characteristics changed	Upline synchronous	603040	acn $\neq$ 0 act = 2, 3 tlc = 1	3-42
CTRL/CHAR/N	Multiple terminal characteristics defined	Upline synchronous	602440	acn $\neq$ 0 act = 2, 3 tlc = 1	3-43
CTRL/CHAR/R	Define multiple terminal characteristics	Downline synchronous	602040	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 1	3-41
CTRL/DEF/R	Redefine terminal characteristic	Downline synchronous	602020	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 6	3-40
CTRL/RTC/A	Bad value in request terminal characteristics supervisory message	Upline synchronous	603044	acn $\neq$ 0 act = 2, 3 tlc = 1	3-45
CTRL/RTC/R	Request current value of terminal characteristics	Downline synchronous	602044	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 1	3-44
CTRL/TCD/R	Terminal characteristics definitions	Upline synchronous	602050	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 1	3-46
DC/CICT/R	Change application character type of connection input	Downline asynchronous	604000	acn = 0 act = 1 tlc = 1	3-35

TABLE 3-1. LEGAL SUPERVISORY MESSAGES (Contd)

Message Mnemonic	Message Meaning	Type	Octal Equivalent of Bits 59 thru 42 of First Text Area Word†	Block Header Fields	Figure Number Defining Message
DC/TRU/R	Truncate upline block	Downline asynchronous	604004	acn = 0 act = 1 tlc = 1	3-37
ERR/LGL/R	Logical error	Upline asynchronous	410004	acn = 0 act = 1 tlc $\geq$ 3	3-57
FC/ACK/R	Output block delivered	Upline asynchronous	406010	acn = 0 act = 1 tlc = 1	3-24
FC/BRK/R	Connection processing interrupted by break	Upline asynchronous	406000	acn = 0 act = 1 tlc = 1	3-27
FC/INACT/R	Connection inactive	Upline asynchronous	406020	acn = 0 act = 1 tlc = 1	3-15
FC/INIT/N	Application ready for connection processing (connection initialized)	Downline asynchronous	406434	acn = 0 act = 1 tlc = 1	3-7
FC/INIT/R	NAM ready for connection processing (connection initialized)	Upline asynchronous	406034	acn = 0 act = 1 tlc = 1	3-6
FC/NAK/R	Output block not delivered	Upline asynchronous	406014	acn = 0 act = 1 tlc = 1	3-25
FC/RST/R	Reset connection	Downline asynchronous	406004	acn = 0 act = 1 tlc = 1	3-28
HOP/DB/R††	Activate debug code	Upline asynchronous	640070	acn = 0 act = 1 tlc = 1	3-47
HOP/DE/R††	Turn off debug code	Upline asynchronous	640074	acn = 0 act = 1 tlc = 1	3-48
HOP/DU/R††	Dump field length	Upline asynchronous	640011	acn = 0 act = 1 tlc = 1	3-49
HOP/NOTR/R††	Turn off AIP tracing	Upline asynchronous	640014	acn = 0 act = 1 tlc = 1	3-51
HOP/REL/R††	Release debug log file	Upline asynchronous	640064	acn = 0 act = 1 tlc = 1	3-52
HOP/RS/R††	Restart statistics gathering	Upline asynchronous	640040	acn = 0 act = 1 tlc = 1	3-53

TABLE 3-1. LEGAL SUPERVISORY MESSAGES (Contd)

Message Mnemonic	Message Meaning	Type	Octal Equivalent of Bits 59 thru 42 of First Text Area Word†	Block Header Fields	Figure Number Defining Message
HOP/TRACE/R††	Turn on AIP tracing	Upline asynchronous	640010	acn = 0 act = 1 tlc = 1	3-50
INTR/APP/R	Application interrupt request	Downline asynchronous	400010	acn = 0 act = 1 tlc = 1	3-29
INTR/RSP/R	Interrupt response	Downline or Upline asynchronous	400004	acn = 0 act = 1 tlc = 0	3-30
INTR/USR/R	User interrupt request	Upline asynchronous	40000_x†††	acn = 0 act = 1 tlc = 1	3-32
LST/FDX/R	Turn on full duplex operation for connections in list	Downline asynchronous	600014	acn = 0 act = 1 tlc = 1	3-23
LST/HDX/R	Turn on half duplex operation for connections in list	Downline asynchronous	600020	acn = 0 act = 1 tlc = 1	3-22
LST/OFF/R	Turn list processing for connection off	Downline asynchronous	600000	acn = 0 act = 1 tlc = 1	3-19
LST/ON/R	Turn list processing for connection on	Downline asynchronous	600004	acn = 0 act = 1 tlc = 1	3-20
LST/SWH/R	Switch application list number of connection	Downline asynchronous	600010	acn = 0 act = 1 tlc = 1	3-21
SHUT/INSD/R	Network shutdown in progress	Upline asynchronous	204030	acn = 0 act = 1 tlc = 1	3-55
TCH/TCHAR/R	Terminal characteristics redefined	Upline asynchronous	310000	acn = 0 act = 1 tlc = 1	3-39
TO/MARK/R	Terminate output marker	Downline synchronous	610000	acn ≠ 0 act = 2, 3 tlc = 2	3-31

†Assumes upper two bits of field following the secondary function code are zero.

††These messages are not currently used. In future releases these messages will be sent by NAM and applications will have to be able to either handle or ignore them. These messages require no response.

†††The setting of bits 43 and 42 depends on the alphabetic character included as the interrupting message from the user.

MANAGING LOGICAL CONNECTIONS

Five messages are used in connection management. These are the CON/ACRQ, CON/REQ, CON/CB, CON/END, and FC/INIT. These messages as well as examples of how they are used in connecting devices to applications, applications to applications, and later terminating these connections are discussed in this subsection.

CONNECTING DEVICES TO APPLICATIONS

After an application program has completed a NETON call, connection-request supervisory messages are sent to the application on behalf of each device seeking connection. Request by request, the application must decide whether to accept or reject the requested connection. Rejection might be necessary, for example, when the application program receives a connection request for a card reader and it does not support batch devices. To respond to a connection-request-message, the application must return one of two similar messages, indicating that the application is either rejecting or accepting the connection request. Figure 3-2 shows the common message sequences in the connection establishment process.

In this figure, arrows indicate the direction of transmission of each message. The general term Network Access Method (NAM) indicates the network software sending or receiving the message, regardless of the software module actually involved.

An application program cannot initiate a connection to a terminal. The connection-request supervisory message shown in figure 3-3 can only be an incoming asynchronous message. The application program's first action in processing a terminal-to-application connection sequence is to issue the asynchronous connection-accepted supervisory message shown in figure 3-4, or the connection-rejected message shown in figure 3-5.

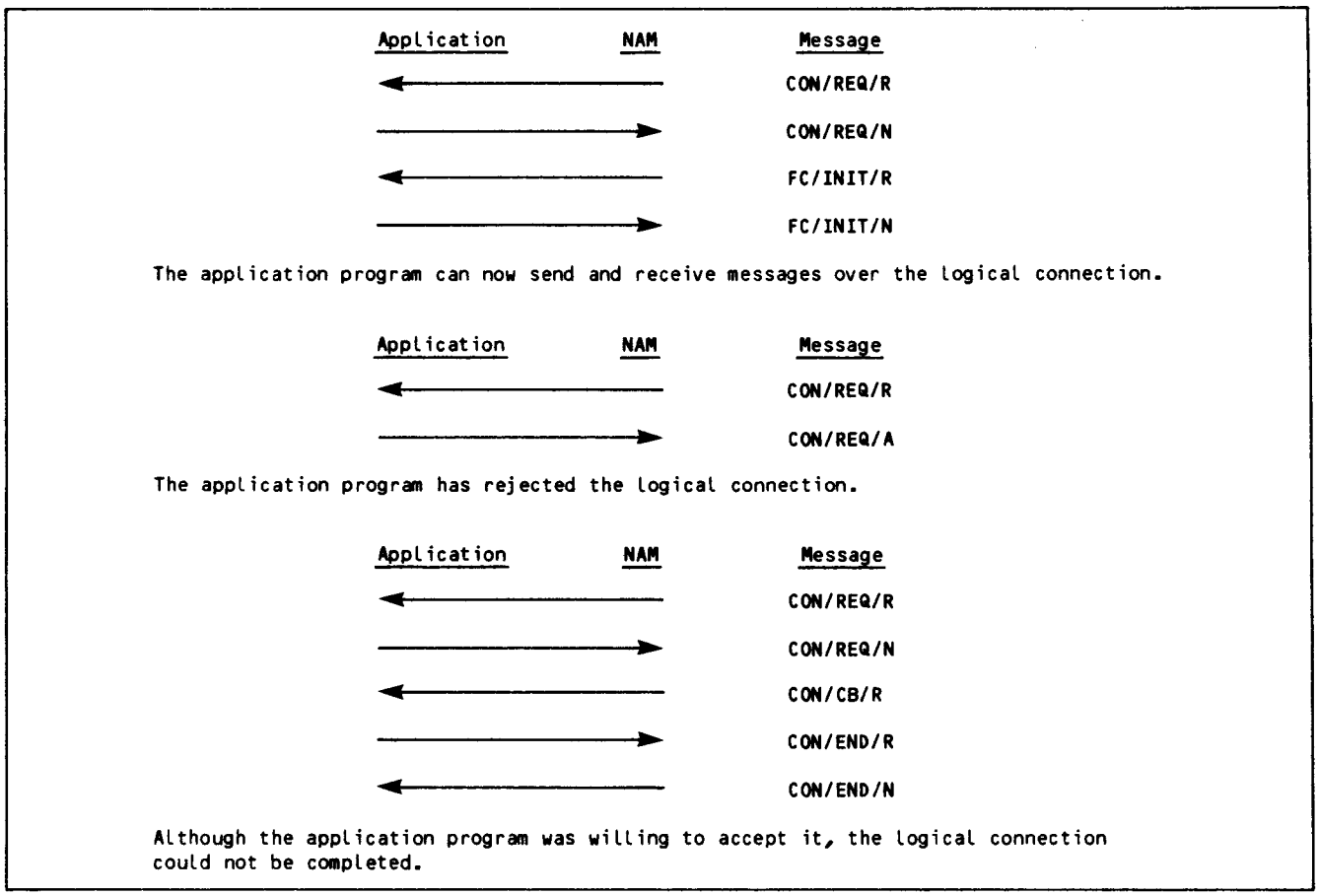
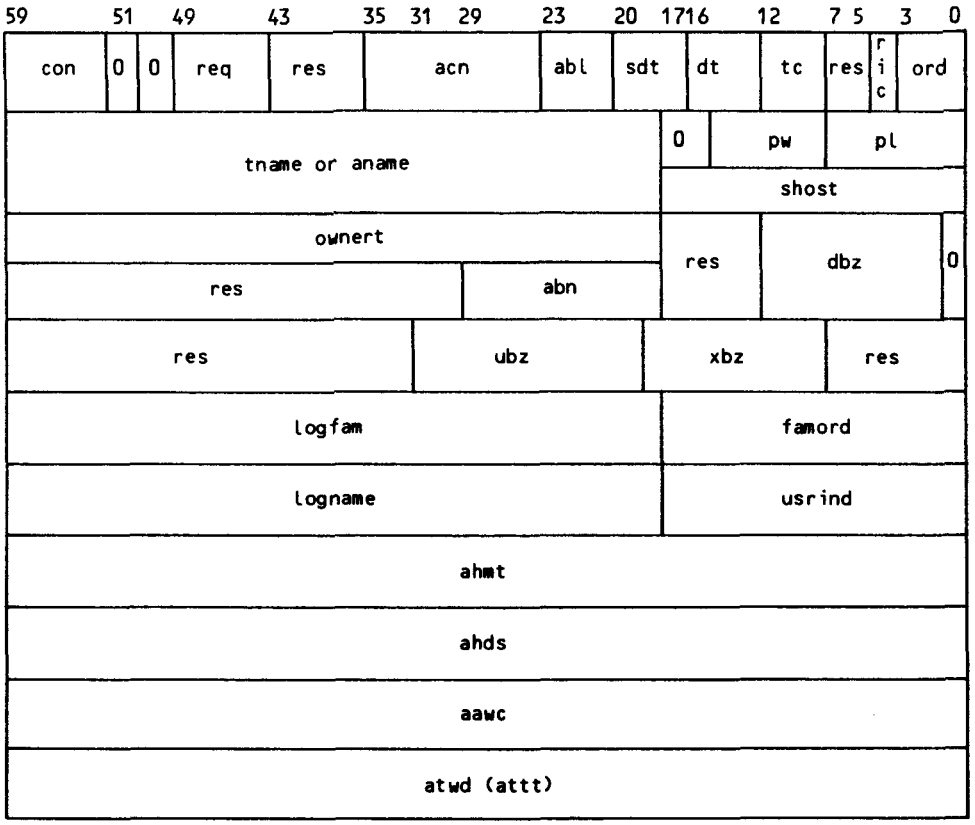


Figure 3-2. Device-to-Application Connection Message Sequence



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of reserved symbol CON.

req — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol REQ.

res — Reserved by CDC. Reserved fields must be equal to zero.

acn — Application connection number assigned to this logical connection, if the connection is established; $1 \leq minacn \leq acn \leq maxacn \leq 4095$, where minacn and maxacn are minimum and maximum values established by the application program in its NETON call. (See section 5.) This field can be accessed with the reserved symbol CONACN, as described in section 4.

abl — Application block limit, specifying the maximum number of data or synchronous supervisory message blocks the program can have outstanding (unacknowledged as delivered by the network software) on this connection at any time. This value is established for the terminal involved in the logical connection when the terminal is described in the local configuration file. This field has the range $1 \leq abl \leq 7$. This field can be accessed with the reserved symbol CONABL, as described in section 4.

sdt — Subdevice type.

IF dt=1, this field can have the values:

0 — 029 punch patterns are the default for each job deck

1 — 026 punch patterns are the default for each job deck

If dt=2, this field can have the values:

0 — 64-character ASCII print train

1 — 64-character BCD (CDC scientific) print train

2 — 95-character ASCII print train

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format (Sheet 1 of 4)

IF dt=4, this field can have the values:

0	Instructions must be packed in 6-bit bytes
1	Instructions must be packed in 8-bit bytes

dt — Device type of the terminal device. This field can have the values:

0	Console (interactive terminal)
1	Card reader
2	Line printer
3	Card punch
4	Plotter
5	Application-to-application connection
6	Reserved for CDC use
7	Reserved for installation use
8 thru 11	Reserved for CDC use

Devices with a device type of zero can be serviced as interactive virtual terminals. Devices with device types of 1 through 4 are serviced as batch devices. This field can be accessed with the reserved symbol CONDT, as described in section 4. Applications other than RBF are only allowed to do input/output on these devices if non-CDC supported batch terminals.

tc — Terminal class assigned to the terminal either in the network configuration file or by the terminal operator. The terminal class determines the parameters and ranges valid for redefinition of the device. The device is serviced by the TIP according to the attributes associated with the terminal class. These attributes are discussed in appendix F. The terminal class field can have the values:

1	Archetype terminal for the class is a Teletype Corporation Model 30 Series.
2	Archetype terminal for the class is a CDC 713-10, 751-1, 756.
3	Reserved for CDC use.
4	Archetype terminal for the class is an IBM 2741.
5	Archetype terminal for the class is a Teletype Corporation Model 40-2.
6	Archetype terminal for the class is a Hazeltine 2000, operating as a teletypewriter.
7	Archetype terminal for the class is a CDC 752.
8	Archetype terminal for the class is a Tektronix 4000 Series, operating as a teletypewriter.
9	Archetype terminal for the class is a HASP (post-print) protocol multileaving workstation.
10	Archetype terminal for the class is a CDC 200 User Terminal.
11	Archetype terminal for the class is a CDC 714-30.
12	Archetype terminal for the class is a CDC 711-10.
13	Archetype terminal for the class is a CDC 714-10/20.
14	Archetype terminal for the class is a HASP (pre-print) protocol multileaving workstation.
15	Archetype terminal for the class is a CDC 734.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format (Sheet 2 of 4)

16 Archetype terminal for the class is an IBM 2780.

17 Archetype terminal for the class is an IBM 3780.

18 thru 27 Reserved for CDC use.

28 thru 31 Reserved for installation use.

This field can be accessed with the reserved symbol CONT, as described in section 4.

ric Restricted interactive capability (for consoles only). This field can have the values:

0 Terminal has unrestricted interactive capability.

1 Terminal has restricted interactive capability.

Applications are advised to limit the amount of interactive dialog with a terminal that has restricted interactive capability. Such terminals (for example a 2780 or 3780) in which the console is emulated by a card reader and line printer are not truly interactive. This field can be accessed with the reserved symbol CONR, as described in section 4.

ord Device ordinal, indicating a unique device when more than one device with the same device type is on the same communication line. This field can have the value:

0 All interactive terminals

1 thru 7 Batch devices

The device ordinal is assigned to the device when the device is defined in the network configuration file. The field can be accessed with the reserved symbol CONORD, as described in section 4.

tname Terminal device name, assigned to the device in the network configuration file. This name is one to seven 6-bit display code letters and digits, left-justified with blank fill; the first character is always alphabetic. The terminal device name is the element name used by the network operator to identify the device. This field can be accessed with the reserved symbol CONTNM, as described in section 4.

pw If the device is an interactive device, this field specifies the maximum number of characters in a physical line of input or output, 0 or $20 \leq pw \leq 255$. If the device is a batch card reader or card punch, this field specifies the maximum number of characters in an input or output record. If the device is a batch line printer, this field specifies the maximum number of characters in a line of output, $50 \leq pw \leq 255$. If the device is a plotter, this field specifies the maximum number of character bytes of plotter information in a record of output. Page width of terminals is discussed in appendix F. This field can be accessed with the reserved symbol CONPW, as described in section 4. PW can be assigned in the network configuration file or the user can set console PW from the terminal. Default value depends on terminal clause.

pl Page length of a device, specifying the number of physical lines that constitute a page. The page length is assigned to the terminal either in the network configuration file or by the terminal operator; page length is one of the attributes associated with the terminal class by the TIP, and is discussed in appendix F. This field can have the values 0 or $8 \leq pl \leq 255$ for interactive terminals, but is always 60 for batch devices. This field can be accessed with the reserved symbol CONPL, as described in section 4.

shost Reserved by CDC.

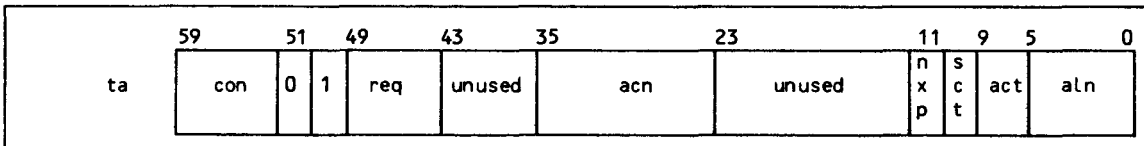
ownert Terminal device name of the controlling console (for batch devices only). This field can be accessed by the reserved symbol CONOWNR, as described in section 4.

dbz Block size in characters for any downline block from the application to NAM. The downline block size is assigned to the device in the network configuration file and is a function of line speed, device type, and terminal class as described in the Network Definition Language reference manual. This field can have the values $1 \leq dbz < 2043$. The values are advisory only. This field can be accessed by the reserved symbol CONDBZ, as described in section 4.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format (Sheet 3 of 4)

abn	Application block number from the application block header of the CON/ACRQ supervisory message if the CON/ACRQ message originated from this application. If the CON/ACRQ message did not originate from this application, the value is zero. This applies to application-to-application connections only. This field can be accessed by the reserved symbol CONAABN, as described in section 4.
ubz	Upline blocking size (in multiples of 100 characters) for a console device. Upline blocking size (in PRUs) of a batch device. This field can be accessed by the reserved symbol CONUBZ, as described in section 4.
xbz	Transmission block size (in characters) of the device. This defines the number of characters in an output transmission block that CCP sends to the terminal. This field can be accessed by the reserved symbol CONXBZ, as described in section 4.
logfam	The NOS family name supplied by the terminal operator during login or by the local configuration file as an automatic login parameter. This family name is one to seven 6-bit display code letters and digits, left-justified with blank fill. This field can be accessed by the reserved symbol CONFAM, as described in section 4.
famord	The NOS family ordinal corresponding to the logfam field contents. This field can be accessed by the reserved symbol CONFO, as described in section 4.
logname	The NOS user name supplied by the terminal operator during login or by the local configuration file as an automatic login parameter. This user name is one to seven 6-bit display code letters, digits, or asterisks, left-justified with blank fill. This field can be accessed by the reserved symbol CONUSE, as described in section 4.
usrind	The NOS user index corresponding to the logname field contents. This field can be accessed by the reserved symbol CONUI, as described in section 4.
ahmt	User validation control word defined in the NOS validation file. This field can be accessed by the reserved symbol CONAHMT, as described in section 4.
ahds	User validation control word defined in the NOS validation file. This field can be accessed by the reserved symbol CONAHDS, as described in section 4.
aawc	User validation control word defined in the NOS validation file. This field can be accessed by the reserved symbol CONAAWC as described in section 4.
atwd(attt)	User validation control word defined in the NOS validation file. This field can be accessed by the reserved symbol CONATWD, as described in section 4.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format (Sheet 4 of 4)



ta	Symbolic address of the application program's text area from which this asynchronous super-supervisory message is sent.
con	Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CON.
req	Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol REQ.
acn	Application connection number assigned by the network software to this end of the logical connection being established with the terminal. The value placed in this field must be the value used in the CON/REQ/R message to which this message is a response. This field can be accessed with the reserved symbol CONACN, as described in section 4.
nxp	No transparent input flag. This field can have the values: 0 Allow transparent input 1 Transparent input not allowed

Figure 3-4. Connection-Accepted (CON/REQ/N) Supervisory Message Format (Sheet 1 of 2)

The change-input-character-type supervisory message, described later in this section, permits an application to change to or from allowing transparent mode input. If transparent input is not allowed any transparent input destined for the application will be discarded. This field can be accessed with the reserved symbol DCNXP, as described in section 4.

sct — Synchronous supervisory message input character type. This field can have the values:

- 0 Application character type 2
- 1 Application character type 3

To indicate the input character type of synchronous supervisory messages the application is willing to accept. The change-input-character-type supervisory message, described later in this section, allows an application to change the input character type of synchronous supervisory messages which the application will receive. This field can be accessed by the reserved symbol DCSCT, as described in section 4.

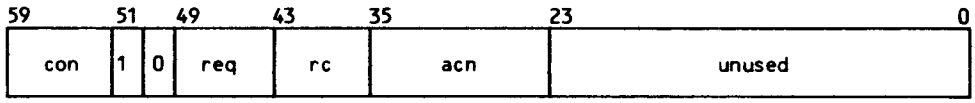
act — Application input character type, specifying the form of character byte packing that the application program requires for input data blocks from the logical connection. This field can have the values:

- 1 60-bit words. Can be used for application-to-application connections within a host. Cannot be used for terminal-to-application connections.
- 2 8-bit characters in 8-bit bytes, packed 7.5 characters per central memory word; if the input is not transparent mode, the ASCII character set described in table A-2 is used.
- 3 8-bit characters in 12-bit bytes, packed 5 characters per central memory word, right-justified with zero fill within each byte; if the input is not transparent mode, the ASCII character set described in table A-2 is used.
- 4 6-bit display coded characters in 6-bit bytes, packed 10 characters per central memory word; the characters used are the ASCII set of CDC characters described in table A-1.

The act value declared applies only to input on the connection and can be changed by a DC/CICT/R supervisory message at any time during the existence of this logical connection. This field can be accessed with the reserved symbol CONACT, as described in section 4.

aln — Application list number assigned by the application program to this logical connection; $0 \leq aln \leq 63$. This field can be accessed with the reserved symbol CONALN, as described in section 4.

Figure 3-4. Connection-Accepted (CON/REQ/N) Supervisory Message Format (Sheet 2 of 2)



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CON.

req — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol REQ.

rc — Reason code, specifying the reason the application program is refusing to complete the connection. This field is ignored. This field can be accessed with the reserved symbol RC, as described in section 4.

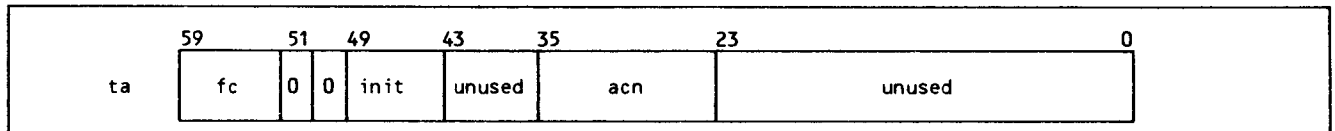
acn — Application connection number assigned by the network software to this end of the logical connection being rejected. The value placed in this field must be the value used in the CON/REQ/R message to which this message is a response. Upon receipt of this message, the network software can reuse this application connection number for a different logical connection with the same program. This field can be accessed with the reserved symbol CONACN, as described in section 4.

Figure 3-5. Connection-Rejected (CON/REQ/A) Supervisory Message Format

If the application program accepts the connection (assuming that no change has occurred in the status of the requesting terminal), the network software informs the application program that the connection is ready for data transmission. This is done by sending the asynchronous initialized-connection message shown in figure 3-6 upline to the application program. If conditions have not changed and the application program can still service the connection, it responds by issuing the connection-initialized message shown in figure 3-7. Data transmission on the logical connection can then begin. After the network software receives the connection-initialized message, the application program can send output to console devices or wait for input from them. An application program cannot send or receive any supervisory messages or data messages on a connection until connection initialization processing has been completed.

If the application program rejects the connection, no further action by the program or the network software occurs. If the application program accepts the connection but the network software cannot initialize the connection, the asynchronous connection-broken supervisory message shown in figure 3-8 is sent to the application program. This connection-broken message requires the application program to respond by issuing an end-connection asynchronous message, as shown in figure 3-9. The network software finishes this sequence by responding with the connection-ended asynchronous supervisory message shown in figure 3-10.

If the application program does not follow these message sequences, a logical-error asynchronous supervisory message is issued to the program. This message is discussed at the end of this section.



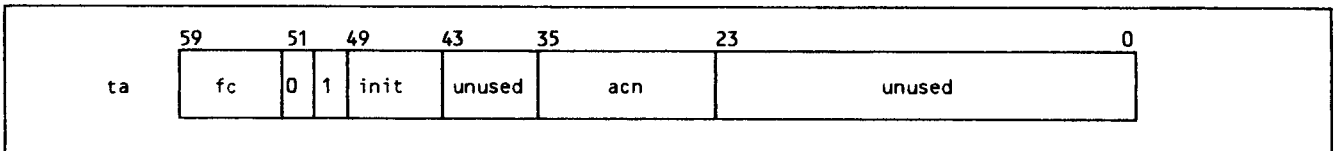
ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

init — Secondary function code 7. This field can be accessed with the reserved symbol SFC, as defined in section 4. Its value is defined as the value of the reserved symbol INIT.

acn — Application connection number assigned by the network software to the program end of the logical connection that has been initialized. This value is the same as that used in previous CON/REQ/R and CON/REQ/N messages. This field can be accessed with the reserved symbol FCACN, as described in section 4.

Figure 3-6. Initialized-Connection (FC/INIT/R) Supervisory Message Format



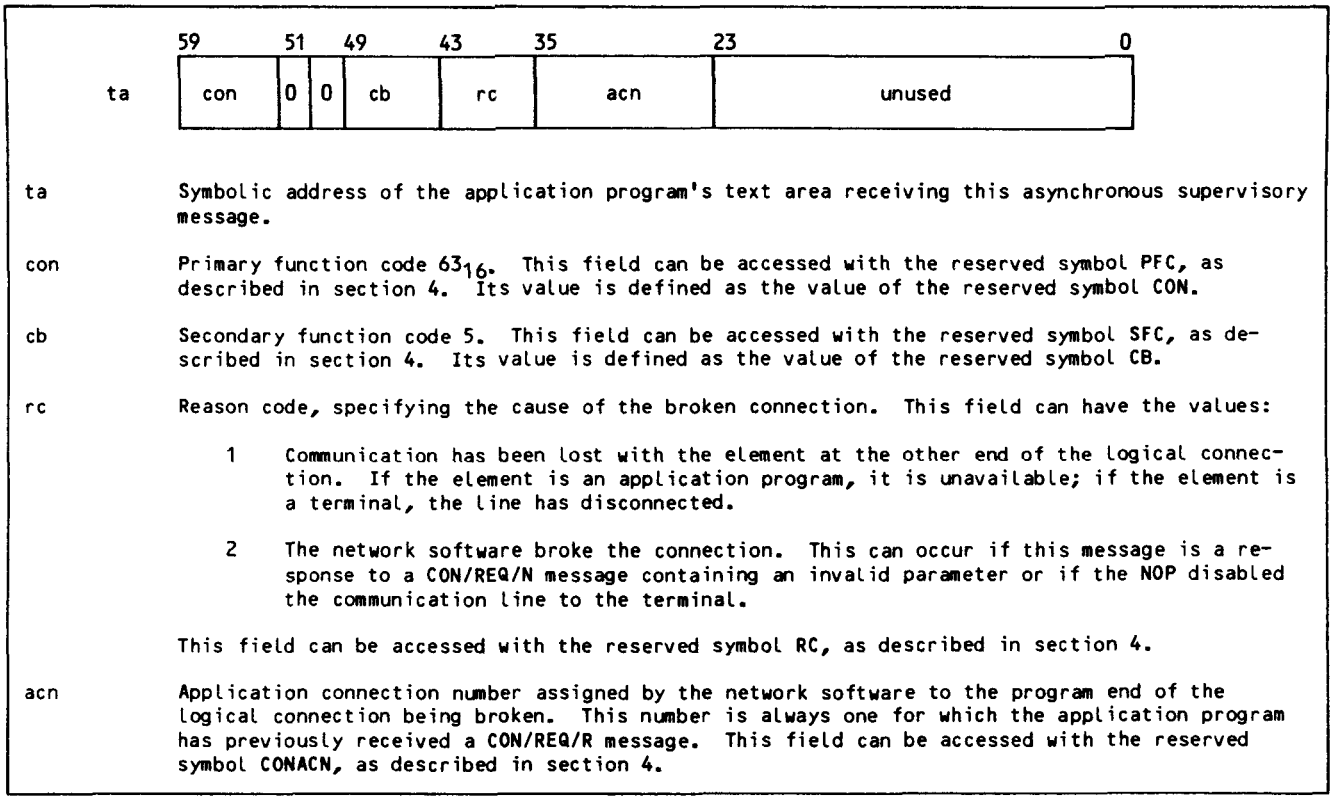
ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

init — Secondary function code 7. This field can be accessed with the reserved symbol SFC, as defined in section 4. Its value is defined as the value of the reserved symbol INIT.

acn — Application connection number assigned by the network software to the program end of the logical connection that has been initialized. This value placed in this field must be the value used in the the FC/INIT/R message to which this message is a response. This field can be accessed with the reserved symbol FCACN, as described in section 4.

Figure 3-7. Connection-Initialized (FC/INIT/N) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CON.

cb — Secondary function code 5. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CB.

rc — Reason code, specifying the cause of the broken connection. This field can have the values:

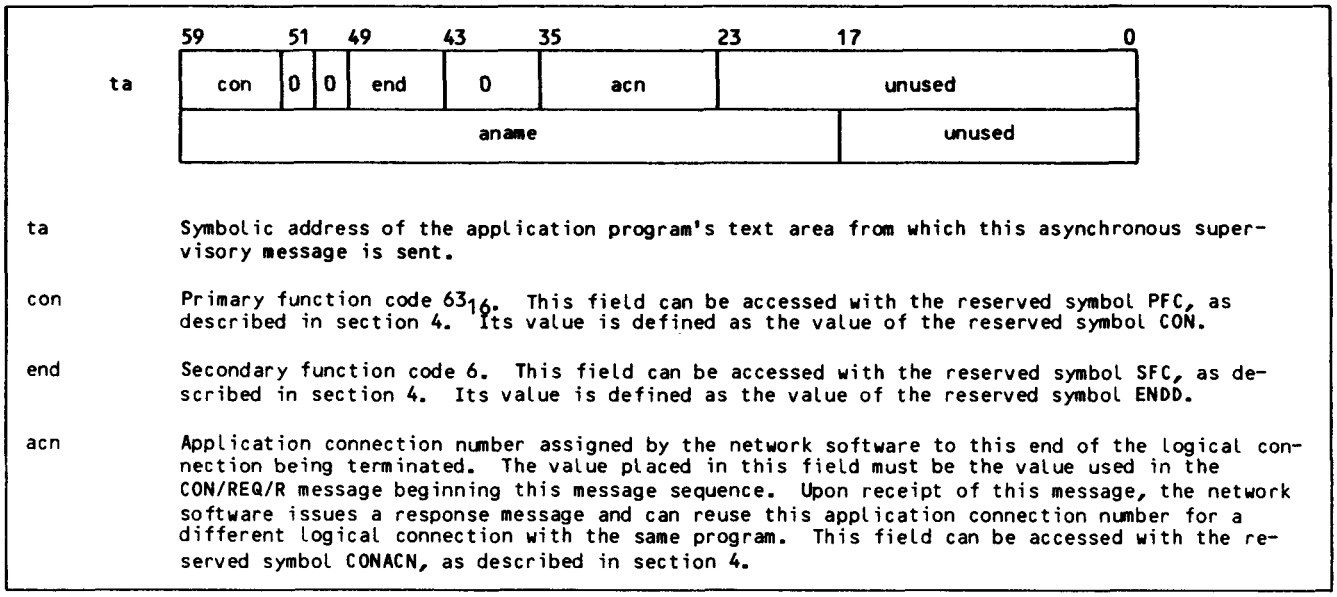
1 Communication has been lost with the element at the other end of the logical connection. If the element is an application program, it is unavailable; if the element is a terminal, the line has disconnected.

2 The network software broke the connection. This can occur if this message is a response to a CON/REQ/N message containing an invalid parameter or if the NOP disabled the communication line to the terminal.

This field can be accessed with the reserved symbol RC, as described in section 4.

acn — Application connection number assigned by the network software to the program end of the logical connection being broken. This number is always one for which the application program has previously received a CON/REQ/R message. This field can be accessed with the reserved symbol CONACN, as described in section 4.

Figure 3-8. Connection-Broken (CON/CB/R) Supervisory Message Format



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CON.

end — Secondary function code 6. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ENDD.

acn — Application connection number assigned by the network software to this end of the logical connection being terminated. The value placed in this field must be the value used in the CON/REQ/R message beginning this message sequence. Upon receipt of this message, the network software issues a response message and can reuse this application connection number for a different logical connection with the same program. This field can be accessed with the reserved symbol CONACN, as described in section 4.

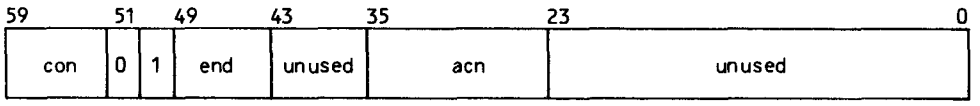
Figure 3-9. End-Connection (CON/END/R) Supervisory Message Format, Connection Establishment Sequences (Sheet 1 of 2)

aname — Name of next application, 1 to 7 characters consisting of letters or digits only with a leading alpha character, left-justified and blank filled within the field. This field is used for terminal-to-application connection only. This field can contain the following:

0	The network software alone determines the next application program that the device is connected to, or disconnects the device if that is an appropriate action.
NVF command	NVF reinitiates the login sequence appropriate for the terminal or causes terminal disconnection. The following commands are valid:
BYE or LOGOUT	Causes the device to be disconnected from the host.
HELLO or LOGIN	Reinitiates login for the device. If dialog is possible and required, the login prompting sequence begins.
Valid application name	The device at the other end of the logical connection is switched (without NVF prompting dialog) to connection with the indicated application, if possible. the name placed in the field must be the element name used to define the referenced application program in the validation file (VALIDUx).

If neither the NVF or valid application name option is used, the value placed in the field must be one to seven 6-bit display code characters consisting of letters or numbers only with a leading alpha character, and blank filled.

Figure 3-9. End-Connection (CON/END/R) Supervisory Message Format, Connection Establishment Sequences (Sheet 2 of 2)



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CON.

end — Secondary function code 6. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ENDD.

acn — Application connection number assigned by the network software to the program end of the logical connection that has been terminated by the CON/END/R message to which this message is a response. After issuing this message, the network software can reassign this application connection number to another logical connection with the same program. This field can be accessed with the reserved symbol CONACN, as described in section 4.

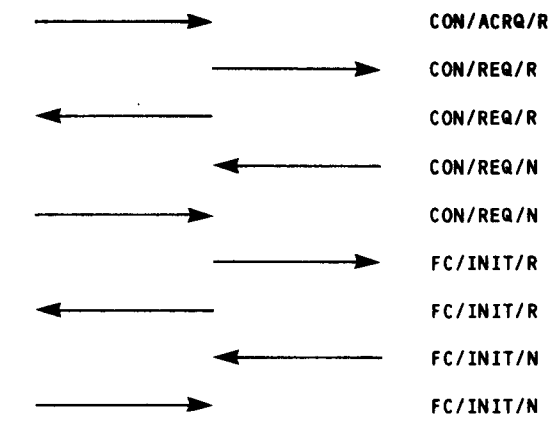
Figure 3-10. Connection-Ended (CON/END/N) Supervisory Message Format

CONNECTING APPLICATIONS TO APPLICATIONS

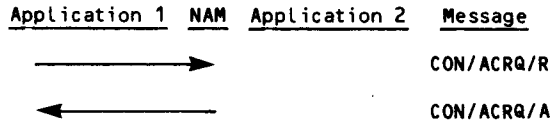
When one application program needs to be connected to another, the first application program sends a supervisory message request to the network software, asking for establishment of a logical connection. Unlike terminal-to-application connections, the network software permits more than one logical connection to exist between two application programs. The only requirements for such connections are that both programs be running, have completed NETON calls (as described in section 5), and have not reached their connection limits.

Figure 3-11 shows the most common message sequences in the process of establishing a connection between two applications.

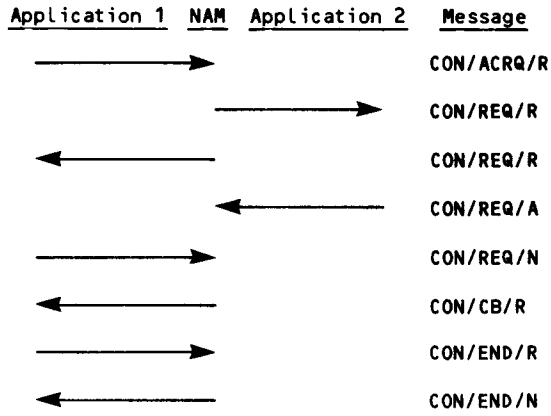
In this figure, arrows indicate the direction of transmission of each message. The general term Network Access Method (NAM) indicates the network software sending or receiving the message, regardless of the software module actually involved.



The requested logical connection is established and enabled for input and output.



Application program 2 is not available. The logical connection is not established.



Application program 2 rejects the logical connection.

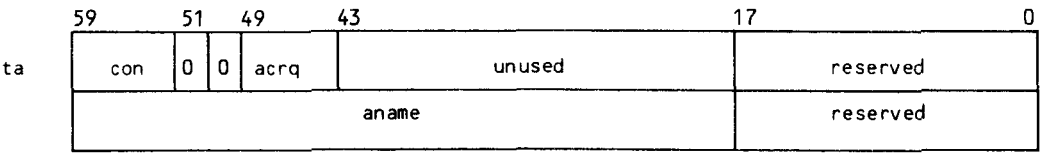
Figure 3-11. Application-to-Application Connection Message Sequences

All three sequences begin when the first application program issues the asynchronous supervisory message shown in figure 3-12. This request-application-connection message causes the network software either to issue the asynchronous application-connection-reject message shown in figure 3-13, or to use a message sequence similar to that used for terminal-to-application connections. If the latter occurs, both application programs receive the form of the asynchronous connection-request supervisory message shown in figure 3-3. Both programs may accept the connection by issuing the form of the connection-accepted asynchronous supervisory message shown in figure 3-4. If so, then both must exchange the initialized-connection and connection-initialized messages of figures 3-6 and 3-7 with the network software before any data can be transmitted on the logical connection.

Neither application program can send or receive any supervisory or data messages on a connection until connection initialization processing has been completed.

If either program cannot complete or service the logical connection, it can reject the connection request by issuing the asynchronous connection-rejected message described in figure 3-5. When this occurs, the other application program must exchange the connection-broken, end-connection, and connection-ended asynchronous supervisory messages with the network software. No further action is required by the rejecting application program.

If either application program does not follow these message sequences, a logical-error asynchronous supervisory message is issued. This message is discussed at the end of this section.



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

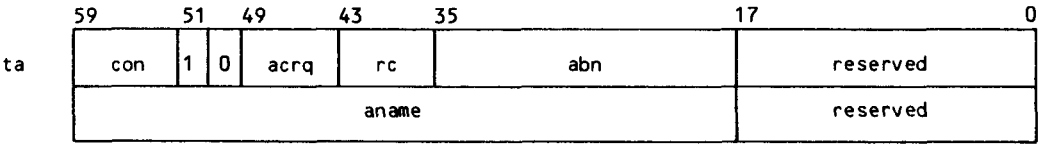
con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of reserved symbol CON.

acrq — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ACRQ.

reserved — Reserved by CDC. Reserved fields must be equal to zero.

aname — Name of the application program with which this program wishes a logical connection to be established. This name can be one to seven 6-bit display coded letters and digits, left-justified with blank fill; the first character is always alphabetic. The name placed in this field must be the element name used to define the requested application program in the network configuration file. This field can be accessed with the reserved symbol CONANM, as described in section 4.

Figure 3-12. Request-Application-Connection (CON/ACRQ/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

con — Primary function code 63_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of reserved symbol CON.

acrq — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ACRQ.

rc — Reason code, specifying the cause for rejecting the connection request. This field can have the values:

- 1 — Call rejected by destination application.
- 2 — A network shutdown is in progress and no additional connections can be established.
- 9 — The receiver application is temporarily not available.
- 10 — The requestor application is temporarily not allowed to make application-to-application connections.
- 12 — Maximum retries reached.

This field can be accessed with the reserved symbol RC, as described in section 4.

abn — Application block number from the application block header of the CON/ACRQ/R supervisory message of the requestor application. This field can be accessed by the reserved symbol CONABN, as described in section 4.

aname — Name of the application program with which connection was requested. This field will always be the same as the aname field in the CON/ACRQ/R message to which this message is a response. This field can be accessed with the reserved symbol CONANM, as described in section 4.

reserved — Reserved by CDC. Reserved fields must be equal to zero.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format

A logical connection established between two application programs does not necessarily have the same application connection number for both applications. The network software assigns the application connection number to each end of the logical connection independently. The application connection number is unique within all connections of each application program, so that the same logical connection can have, for example, an acn parameter of 2 for application program A (which accepted one previous connection) but an acn parameter of 4 for application program B (which accepted three previous connections).

MONITORING CONNECTIONS

As soon as a logical connection is completely initialized by the network software and an application program, the network software begins incrementing an inactivity timer. Each time a data block or synchronous supervisory message is transmitted on the logical connection, this inactivity timer is reset to zero. Any time 10 minutes elapse without any message transmission on a logical connection, the network software uses one of the supervisory message sequences shown in figure 3-14 to inform the application program of the condition.

The connection monitoring sequence consists of the asynchronous inactive-connection message shown in figure 3-15. This message is advisory only; no response is required from the application program. The network software automatically resets the inactivity timer to zero as soon as the message is issued.

Connection monitoring is not performed for logical connections to batch devices.

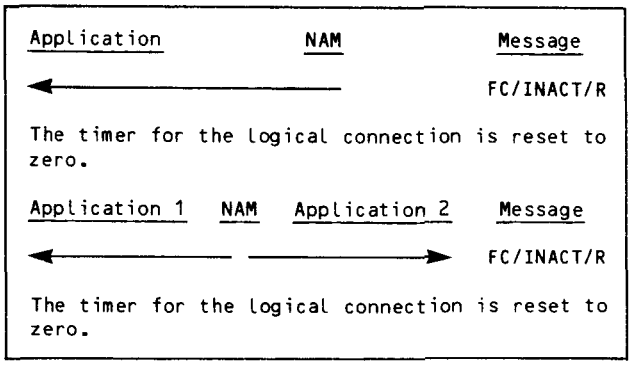
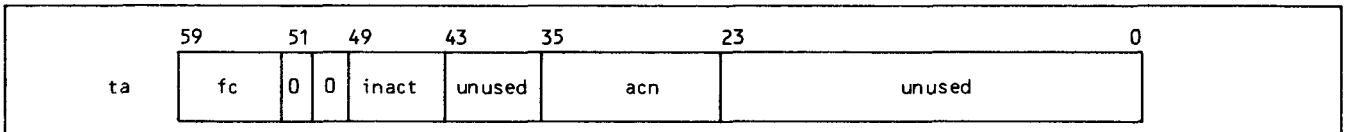


Figure 3-14. Connection Monitoring Message Sequences

TERMINATING CONNECTIONS

A logical connection can be terminated any time after establishment of it begins. This disconnection can be initiated by an application program or by the network software. These two possibilities have separate corresponding supervisory message sequences, as shown in figure 3-16.

A logical connection termination is indicated by the network whenever a condition arises that is not caused by the application program. Such conditions include hardware failure, a dialup line being disconnected without a formal logout by a terminal operator, failure of another (connected) application program, and so forth. The general case of this is shown by the second message sequence in the figure, a sequence already encountered as part of the connection establishment sequences discussed earlier in this section.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

inact — Secondary function code 4. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol INACT.

acn — Application connection number assigned by the network software to the program end of the logical connection reported as inactive. The value in this field is always nonzero and is the value used in an FC/INIT/N message processed by the application program. This field can be accessed with the reserved symbol FCACN, as described in section 4.

Figure 3-15. Inactive-Connection (FC/INACT/R) Supervisory Message Format

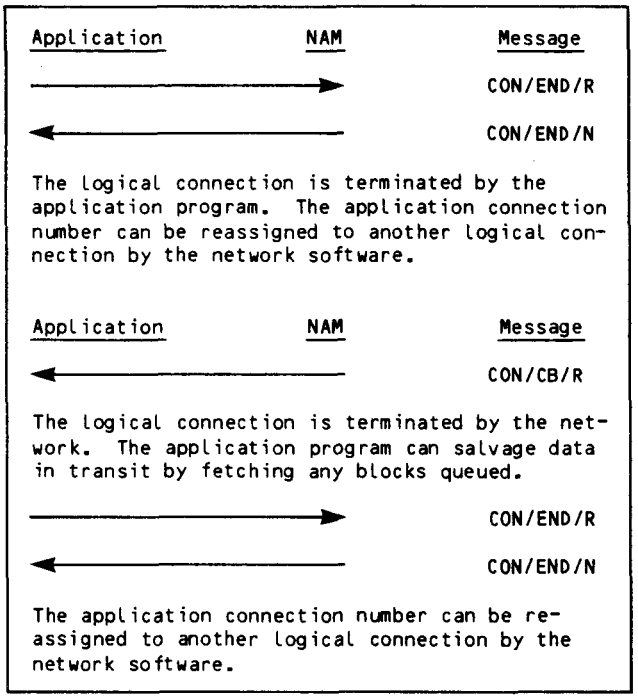


Figure 3-16. Connection Termination Message Sequences

The sequence begins when the network software sends the connection-broken message of figure 3-8 to the application program. The network software discards any data blocks or synchronous supervisory messages sent by the application program on the connection between the time this asynchronous supervisory message is queued and the time it is processed by the application program. When the application program receives this message, it can still fetch any upline blocks queued on the logical connection. As soon as it has fetched all outstanding blocks, the application program must issue an end-connection message of the form shown in figure 3-9. The network software responds with the asynchronous connection-ended message described in figure 3-10. The application connection number of the terminated logical connection then becomes available for use with another logical connection.

Application-initiated termination of a logical connection occurs whenever the application program processes a terminal operator's request to end connection, or in any other situation where the application program has finished exchanging blocks over the logical connection. The message sequence is the first one shown in figure 3-16. This sequence begins when the application program issues an asynchronous end-connection supervisory message.

The format of the end-connection message is described in figure 3-9. This message permits the application program to influence connection switching or disconnection processing performed for the device after it is disconnected from the application program. The effects of this end-connection message vary according to the aname field contents and whether the device is a passive (batch) or interactive (console) device.

The contents of aname for passive devices is irrelevant. NVF will always logout a passive device if it's connection is ended.

When a zero aname parameter is used an interactive device is prompted for the name of the next program the device should be connected to, unless the user is allowed access only to the disconnected application program. In this instance, the device's logical connection is processed by NVF as if an aname value of BYE or LOGOUT was specified.

When a valid application name is used in the aname field for a console connection, the connection is disposed of in one of two ways. If the specified application program is available and the login user name of the console is allowed access to it, the console connection is switched directly to the new application program. This switch is performed without dialog between NVF and a console operator. The network software performs the switch by sending a connection-request supervisory message for the console to the specified application program.

If the specified application program is not available or the login user name does not permit the terminal to access it, the terminal connection is not switched. In this case, an interactive terminal is informed of the condition with the message APPLICATION NOT PRESENT or USER ACCESS NOT POSSIBLE - CONTACT NETWORK ADMINISTRATOR. The terminal is then prompted for another application program name, unless the terminal was configured for a full automatic login procedure and the user name in that procedure validates access only to the disconnected application program. In this instance, the terminal's logical connection is processed by NVF as if an aname value of BYE or LOGOUT was specified.

When an NVF command is used in the aname field, disconnection processing depends on the command used and whether the terminal is a batch or interactive terminal. The HELLO or LOGIN command causes NVF to initiate a manual login dialog with an interactive terminal.

The BYE or LOGOUT command causes NVF to attempt disconnection of the terminal from the data communication network. NVF requests the network software to perform disconnection processing for batch and interactive terminals.

When passive devices are logged out, they will be reconnected to the same host application as its owning console if the console has not logged out.

On dialup lines, interactive terminals are assigned to a disconnection queue. When all interactive terminals on the dialup line are assigned to the disconnection queue, a timer for the line is started. When the timer for the line expires, the dialup line is physically disconnected. This disconnection causes physical disconnection of all terminals on the line, including any passive terminals still connected to an application program (the connection is broken from the application program's viewpoint). The network software effectively hangs up the telephone, but the terminals can be reconnected after a new dial-in procedure.

On hardwired lines, no disconnection occurs when all interactive terminals on the hardwired line are timed out. Because the line is not disconnected in this instance, passive terminals still connected to application programs remain connected to those programs.

While a terminal is queued for disconnection, any terminal operator keyboard entry removes the terminal from the disconnection queue and reconnects it to NVF for a new manual login procedure. The data entered is discarded by the network software and therefore can be anything the operator wishes.

MANAGING CONNECTION LISTS

There are five asynchronous supervisory message sequences used for connection list management. Each sequence consists of one message, issued by the application program.

Three of these sequences, as shown in figure 3-17, control list polling and list assignment. The other sequences, shown in figure 3-18, control the duplexing mode used during list processing.

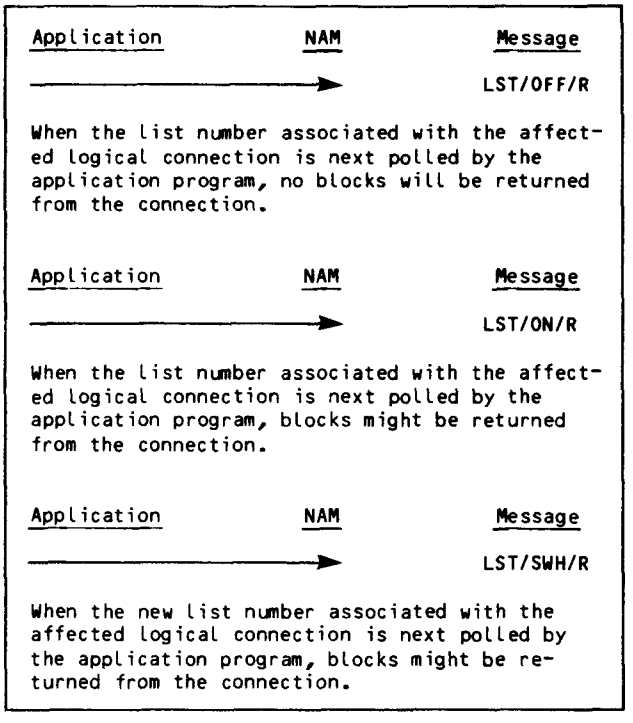


Figure 3-17. Connection List Polling Control Message Sequences

CONTROLLING LIST POLLING

Connection list polling control consists of enabling or disabling the fetching of input blocks from a single logical connection when the list that the connection is assigned to is polled. All connections are initially enabled for list processing without application program action. Each time the application program polls the list number that it has associated with a specific connection, blocks queued from that connection can be returned to the program. If the program requires the list to be polled without returning any blocks queued from the connection, the asynchronous supervisory message shown in figure 3-19 causes the next poll of the list to exclude the connection. This turn-list-processing-off message effectively disables list processing for the connection. This message is not acknowledged by the network software and remains in effect until canceled by the asynchronous turn-list-processing-on message shown in figure 3-20.

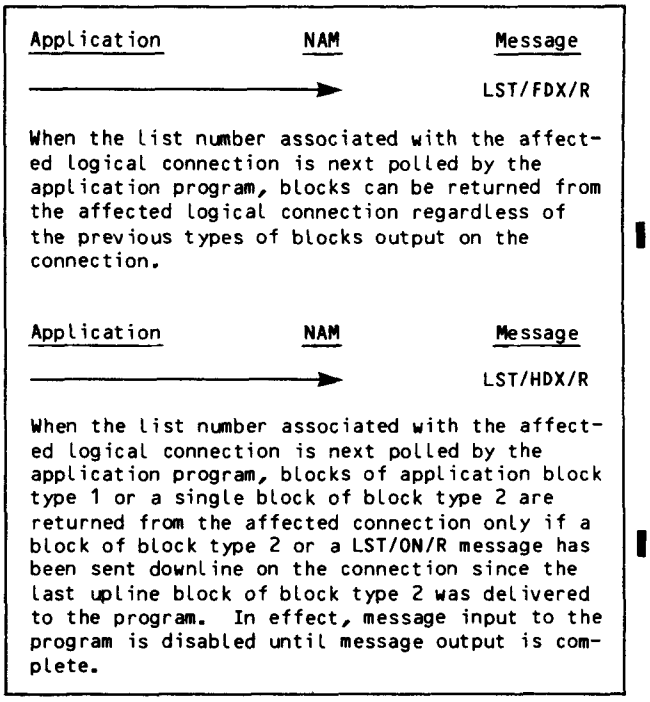
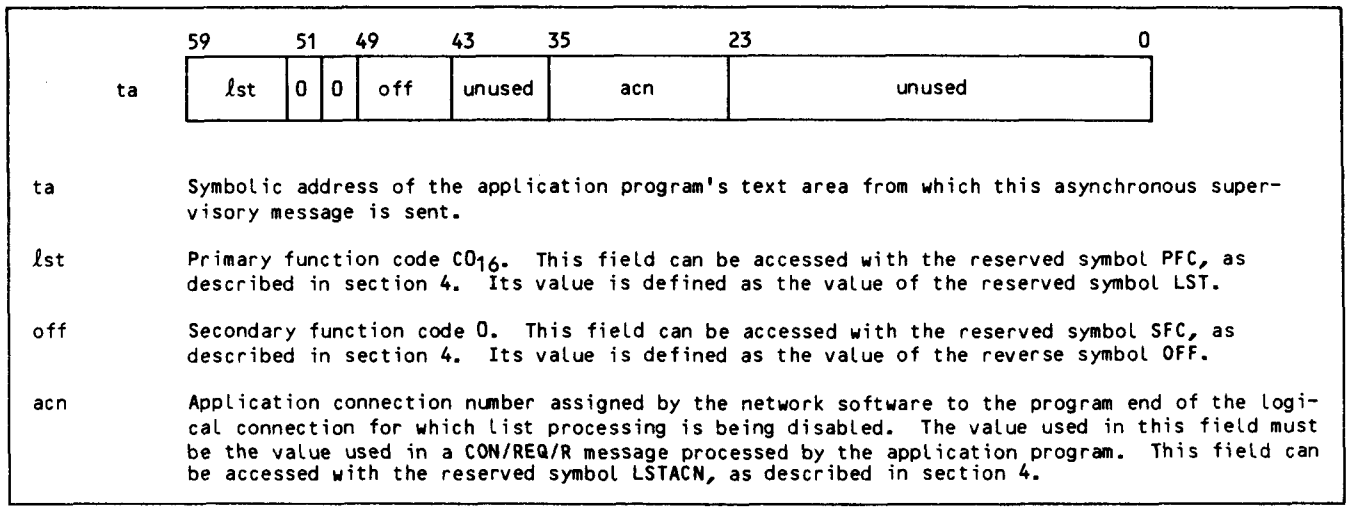


Figure 3-18. Connection List Duplexing Message Sequences

The turn-list-processing-on message is issued by the application program to enable list processing and input for a specific connection. This message causes the next poll of the list number associated with the indicated connection to include the connection's data block queue. The network software does not acknowledge this message. If the message is issued when list processing already has been enabled for the connection, no error occurs. The message remains in effect until canceled by a turn-list-processing-off supervisory message.

Enabling list processing for a logical connection does not cause a queued block to be returned from that connection the next time the connection's list is polled. Connections on a list are searched in a round-robin fashion starting with the connection following the connection from which data was last obtained. Disabled connections are skipped during the polling process; enabled connections and connections in half-duplex mode for which no output has been sent are included.



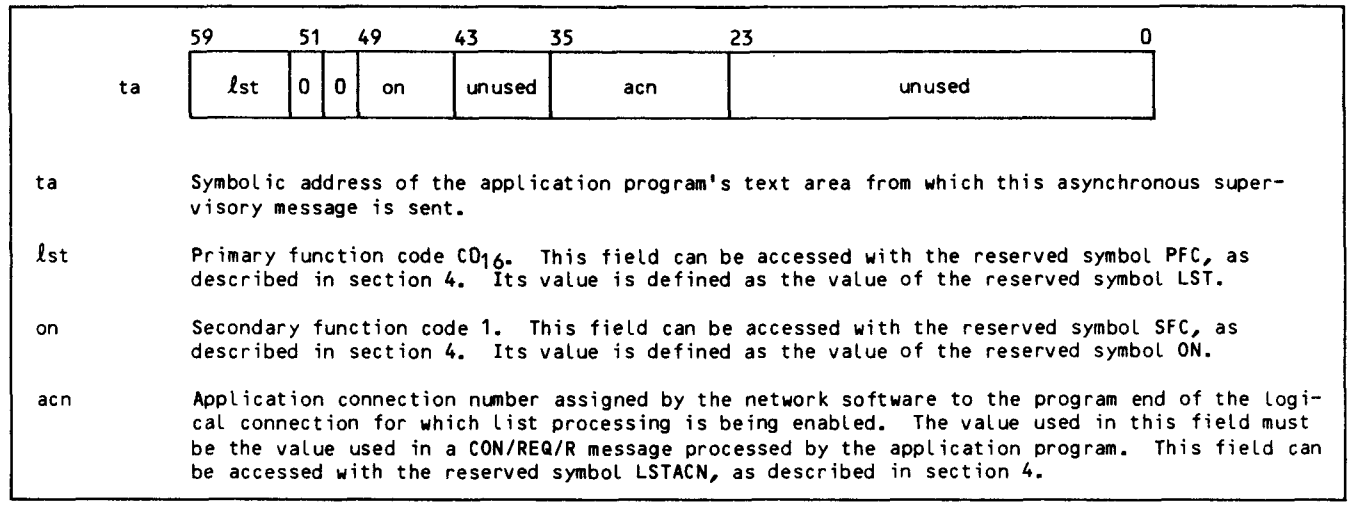
ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

ℓst — Primary function code CO_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol LST.

off — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reverse symbol OFF.

acn — Application connection number assigned by the network software to the program end of the logical connection for which list processing is being disabled. The value used in this field must be the value used in a CON/REQ/R message processed by the application program. This field can be accessed with the reserved symbol LSTACN, as described in section 4.

Figure 3-19. Turn-List-Processing-Off (LST/OFF/R) Supervisory Message Format



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

ℓst — Primary function code CO_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol LST.

on — Secondary function code 1. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ON.

acn — Application connection number assigned by the network software to the program end of the logical connection for which list processing is being enabled. The value used in this field must be the value used in a CON/REQ/R message processed by the application program. This field can be accessed with the reserved symbol LSTACN, as described in section 4.

Figure 3-20. Turn-List-Processing-On (LST/ON/R) Supervisory Message Format

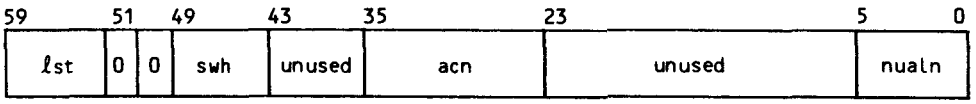
The list number associated with a specific connection is determined by the application program when it accepts the logical connection. This list number can be changed while the connection exists by issuing the change-connection-list supervisory message shown in figure 3-21. The network software does not acknowledge this asynchronous message, but the change is effective at the time of the next poll of the new list number. After the change-connection-list message is issued by the application program, polls of the old list number cannot return blocks queued from the affected connection.

Polling of connection lists is performed through application calls to the AIP routines NETGETL and NETGTFL. These routines are described in section 5.

CONTROLLING LIST DUPLEXING

Upline and downline transmissions on logical connections usually occur in a full duplex mode. In full duplex mode, the number and occurrence of complete upline message blocks is not related in any way to the number or occurrence of downline message blocks. Message input and output is logically independent and can become unsynchronized.

The list processing feature of NAM can be used in conjunction with a set of asynchronous supervisory messages to avoid loss of input and output synchronization on a logical connection. These messages can be used to switch the connection to and from a half duplex mode of input and output.



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

ℓst — Primary function code CO_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol LST.

swh — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol SWH.

acn — Application connection number assigned by the network software to the program end of the logical connection being switched to a new connection list. The value used in this field must be the value used in a CON/REQ/R message processed by the application program. This field can be accessed with the reserved symbol LSTACN, as described in section 4.

nualn — The number of the new connection list to which the logical connection is reassigned; $0 \leq$ nualn ≤ 63. This field can be accessed with the reserved symbol LSTALN, as described in section 4.

Figure 3-21. Change-Connection-List (LST/SWH/R) Supervisory Message Format

In half duplex mode, delivery of an upline block of block type 2 turns off additional list processing for the connection until a downline block of block type 2 or a LST/ON/R message is sent on the same connection. In effect, application program input obtained through NETGETL or NETGTFL calls must alternate with output for the connection, because no other sequence of input and output is possible using those calls.

An application program begins network access with its AIP list processing code automatically enabled for full-duplex operation of all logical connections. The program can change a single connection to half-duplex operation at any time during network access by issuing the asynchronous supervisory message shown in figure 3-22, with the appropriate application connection number included in the acn field. Alternatively, the program can change all existing and any future connections by issuing the same supervisory message with an acn field value of zero. There is no response to either form of this message.

When half-duplex operation begins for a connection, the connection is initially enabled or disabled for normal list processing, depending on the setting of the reserved symbol LSTDIS in the LST/HDX/R supervisory message shown in figure 3-22. If LSTDIS is set to zero, then the connection is initially enabled for normal list processing via NETGETL or NETGTFL calls. When such a call returns a block of application block type 2 (identifying the last block of an upline message), NETGETL or NETGTFL calls disable the connection for subsequent list processing.

Use of the turn-on-half-duplex-list-processing message has no effect on use of the turn-list-processing-off or turn-list-processing-on messages. The effects of the latter messages take precedence over the mode of duplexing operation in effect for a given connection. In addition, the turn-list-processing-on message enables the connection for input, even if no output has been sent.

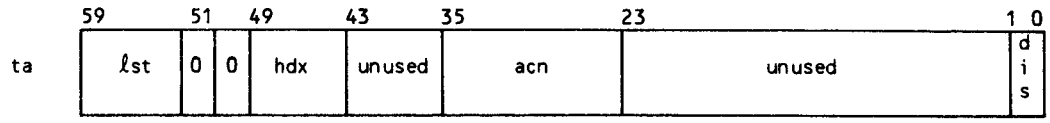
An application program can change a single connection back to full-duplex operation at any time during network access by issuing the asynchronous supervisory message shown in figure 3-23, with the appropriate application connection number included in the acn field. Alternatively, the program can change all existing and any future connections by issuing the same supervisory message with an acn field value of zero. There is no response to either form of this message.

When full duplex operation begins for a connection, the connection is initially enabled for normal list processing via NETGETL or NETGTFL calls. The connection remains enabled until disabled by the previously described turn-list-processing-off supervisory message. Upline delivery of a data block of application block type 2 has no relationship to downline transmission of a block of the same block type.

Use of the turn-on-full-duplex-list-processing message has no effect on use of the turn-list-processing-off or turn-list-processing-on messages. The effects of the latter messages take precedence over the mode of duplexing operation in effect for a given connection. If a given connection has been disabled for any list processing by a turn-list-processing-off message, it remains disabled after full-duplex operation is turned on for the connection.

If either of the list duplexing control messages is issued for a connection already operating in the requested duplexing mode, the extra message is ignored. If the acn field specified within either message identifies a nonexistent logical connection, a logical-error supervisory message is sent to the application program and the requested change in duplexing operation does not occur.

If either of the list duplexing control messages is issued with an acn field value of zero, the duplexing mode of application connection zero remains unchanged. The asynchronous supervisory message connection is always enabled for full duplex operation on application list zero.



ta — Application program text area from which this asynchronous supervisory message is sent.

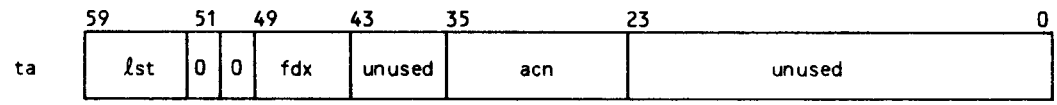
ℓst — Primary funtion code CO_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol LST.

hdx — Secondary function code 4. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol HDX.

acn — Application connection number assigned by the network software to the program end of the logical connection for which half-duplex list processing is being enabled. The value used in this field can be either zero or the value used in a CON/REQ/R message processed by the application program. If acn is zero, all connections are enabled; if acn is nonzero, the specific connection is enabled. This field can be accessed with the reserved symbol LSTACN, as described in section 4.

dis — Disable flag. The value of this flag is set either to 1 if the connection is to be initially disabled for normal list processing or to 0 if the connection is to be initially enabled for normal list processing. This field can be accessed with the reserved symbol LSTDIS, as described in section 4.

Figure 3-22. Turn-On-Half-Duplex-List-Processing (LST/HDX/R) Supervisory Message Format



ta — Application program text area from which this asynchronous supervisory message is sent.

ℓst — Primary function code CO_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol LST.

fdx — Secondary function code 3. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol FDX.

acn — Application connection number assigned by the network software to the program end of the logical connection for which full-duplex list processing is being enabled. The value used in this field can be either zero or the value used in a CON/REQ/R message processed by the application program. If acn is zero, all connections are enabled; if acn is nonzero, the specified connection is enabled. This field can be accessed with the reserved symbol LSTACN, as described in section 4.

Figure 3-23. Turn-On-Full-Duplex-List-Processing (LST/FDX/R) Supervisory Message Format

CONTROLLING DATA FLOW

Data to and from console connections has its flow controlled at both ends of those connections. Whenever possible, this control is imposed voluntarily by the application program. Conditions outside the network, however, can interfere with data flow. Flow control is therefore also imposed by the network software in reaction to external conditions. When the latter occurs, the application program must compensate for the effect on data flow.

Because the application program is not directly involved in the data exchange on batch device connections, the remaining paragraphs in this subsection do not apply to application-to-batch device connections.

Downline flow control is logically separated from upline flow control. This separates flow control into an input function and an output function.

Downline flow control is implemented through block delivery monitoring mechanisms. These mechanisms involve exchanges of asynchronous supervisory messages, and the application program's adherence to data block transmission conventions.

Upline input flow is controlled by synchronous supervisory messages and by the application program's adherence to data block transmission conventions.

MONITORING DOWNLINE DATA

An application program can send downline blocks along a particular connection much faster than they can be output at the device. Since NAM and CCP must save these extra blocks until they are processed by the device, the extra blocks can cause NAM and CCP to have storage problems. On the other hand, the same application program might be sending blocks along another connection at such a slow rate that its device is under-occupied. Network software provides a set of conventions that allow the application to control the flow of data between itself and its devices for increased efficiency in such cases.

A block limit is established for each logical connection; this parameter indicates how many blocks of data or synchronous supervisory messages an application program can have outstanding on the logical connection at any instant. This block limit is the abl field value included in the connection-request supervisory message. As blocks are delivered to the device, a block-delivered asynchronous supervisory message (figure 3-24) is returned to the application. If the application program's output exceeds the value of the block limit, a logical-error asynchronous supervisory message is returned to the application, together with the reason for the failure, and the last block is discarded by NAM.

The block-delivered supervisory message is used to manage flow control. However, receipt of a block delivered supervisory message does not in all cases guarantee that the data block has reached its destination.

If the application program's output does not exceed the block limit, but for some reason a block is lost or unaccounted for, a block-not-delivered asynchronous supervisory message (figure 3-25) is returned to the application. Neither the block-delivered message nor the block-not-delivered message requires the application program to issue a response or acknowledgment message to NAM.

This protocol allows the application to control downline data flow, as follows:

Define two arrays, K and M.

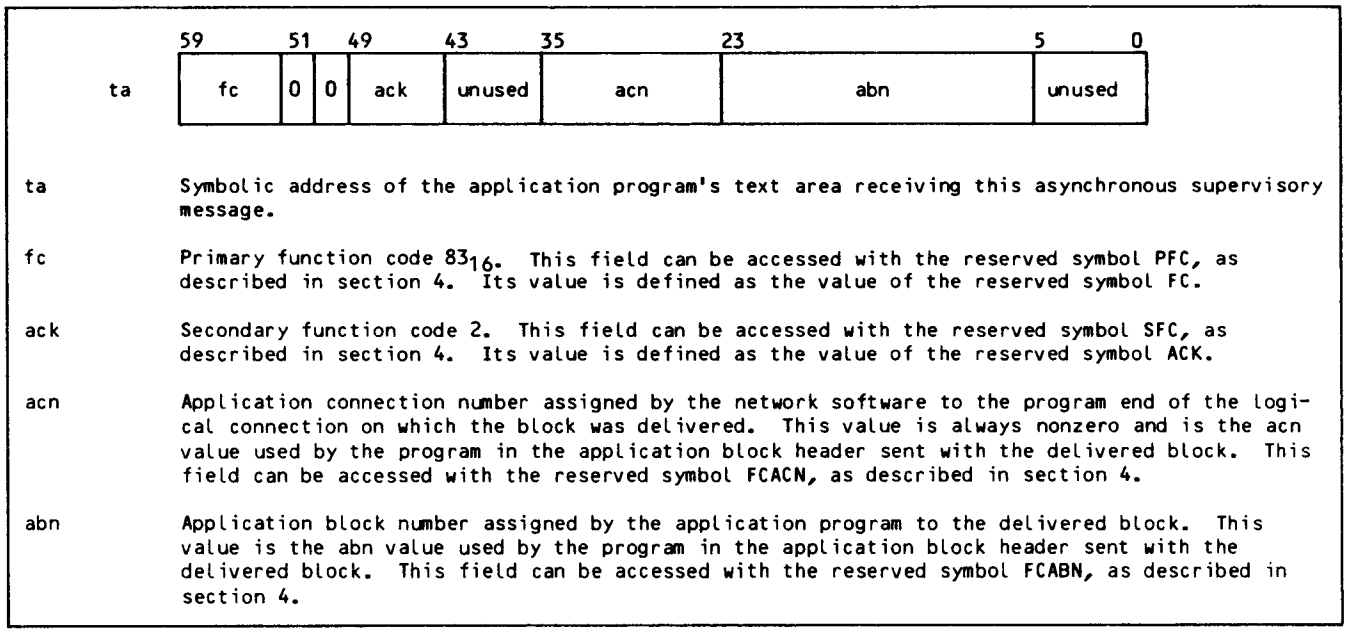
When a connection i is accepted, set K(i)=0 and M(i)=block limit.

Whenever a block-delivered message is received for application connection number i, set K(i)=K(i) - 1.

Whenever a break supervisory message is received, set K(i)=0.

As long as K(i) is less than M(i), set K(i) + 1 and output one block on connection i.

The break supervisory message included in this strategy affects downline traffic on a logical connection. Such a message is sent to the application program whenever a network condition requires downline transmission on the connection to be interrupted.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

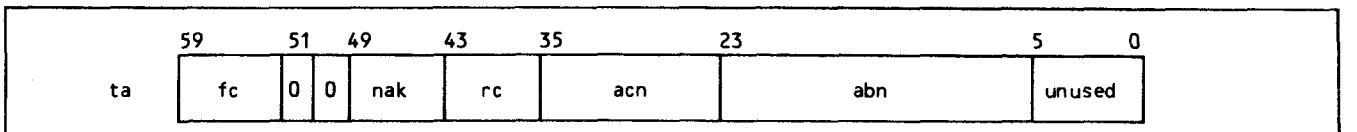
fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

ack — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ACK.

acn — Application connection number assigned by the network software to the program end of the logical connection on which the block was delivered. This value is always nonzero and is the acn value used by the program in the application block header sent with the delivered block. This field can be accessed with the reserved symbol FCACN, as described in section 4.

abn — Application block number assigned by the application program to the delivered block. This value is the abn value used by the program in the application block header sent with the delivered block. This field can be accessed with the reserved symbol FCABN, as described in section 4.

Figure 3-24. Block-Delivered (FC/ACK/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

nak — Secondary function code 3. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol NAK.

rc — Reason code explaining why the block was not delivered. This field can have the value:

1 Network software error caused loss of the block in transit; the block can be retransmitted but might be delivered out of sequence with subsequently transmitted blocks.

This field can be accessed with the reserved symbol RC, as described in section 4.

acn — Application connection number assigned by the network software to the program end of the logical connection on which the block was lost. This value is always nonzero and is the acn value used by the program in the application block header sent with the lost block. This field can be accessed with the reserved symbol FCACN, as described in section 4.

abn — Application block number assigned by the application program to the lost block. This value is the abn value used by the program in the application block header sent with the lost block. This field can be accessed with the reserved symbol FCABN, as described in section 4.

Figure 3-25. Block-Not-Delivered (FC/NAK/R) Supervisory Message Format

The NPU relies on the application program to decide when traffic can be resumed. To prevent ambiguous situations and allow the application program maximum flexibility in reacting to such an interruption, the following sequence of events always occurs (see figure 3-26):

1. Blocks output by the application program but not yet delivered to a device are discarded.
2. A break supervisory message (figure 3-27) is sent to the application program.
3. The application program receives the break supervisory message.
4. For application-to-terminal connections NAM queues a null block with the break-occurred bit set following any data blocks that preceded the occurrence of the break. This null block serves as a marker to identify the point in the data stream where the break occurred.
5. The application program then must issue a reset asynchronous supervisory message, as shown in figure 3-28, as a response to the break message. No response to the reset message is sent by NAM to the application. Normal downline traffic can now resume.

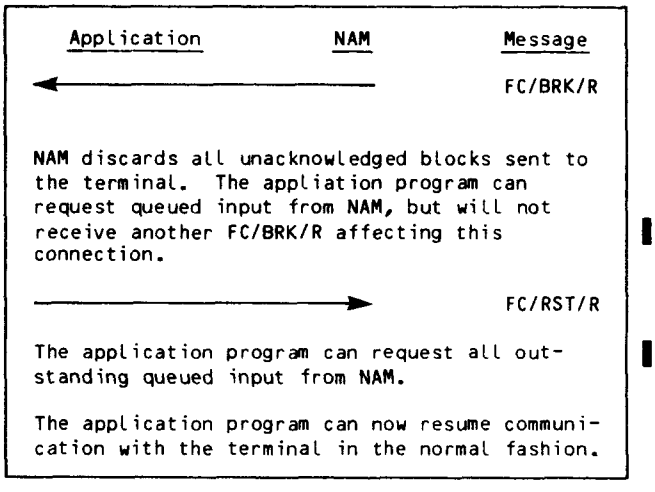
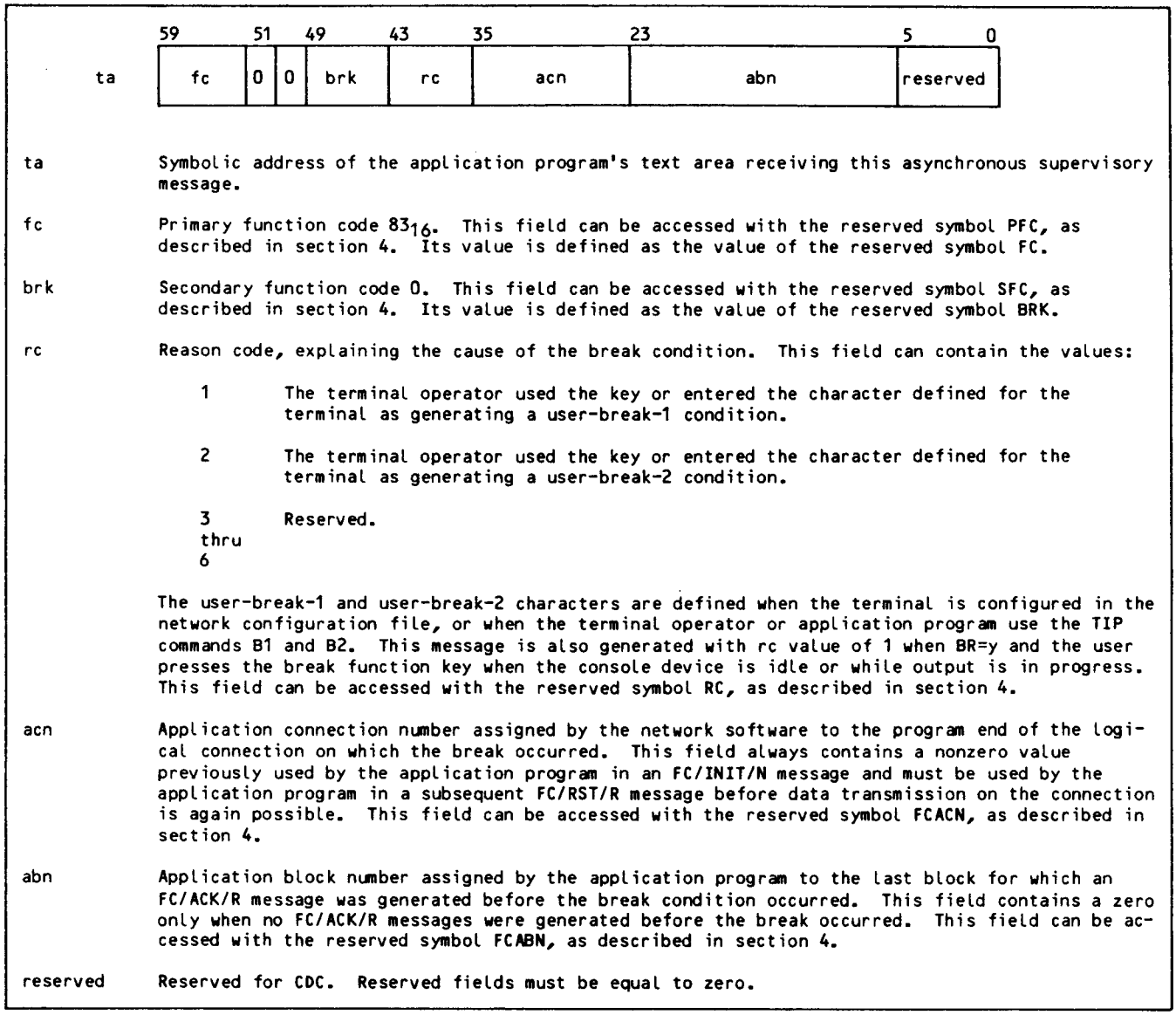


Figure 3-26. Break and Reset Message Sequence

6. The application program can process all pending input by issuing NETGET or NETGETF calls (section 5) on that connection until a null block is received with the break-occurred bit set. The disposition of these blocks is up to the application.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

brk — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol BRK.

rc — Reason code, explaining the cause of the break condition. This field can contain the values:

1	The terminal operator used the key or entered the character defined for the terminal as generating a user-break-1 condition.
2	The terminal operator used the key or entered the character defined for the terminal as generating a user-break-2 condition.
3 thru 6	Reserved.

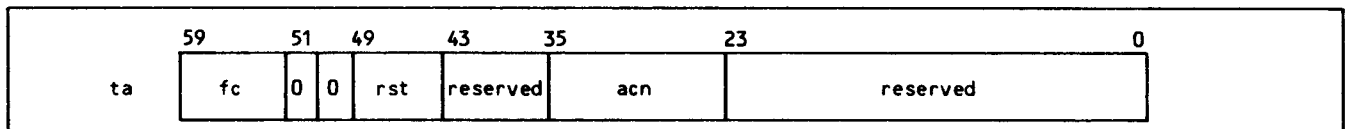
The user-break-1 and user-break-2 characters are defined when the terminal is configured in the network configuration file, or when the terminal operator or application program use the TIP commands B1 and B2. This message is also generated with rc value of 1 when BR=y and the user presses the break function key when the console device is idle or while output is in progress. This field can be accessed with the reserved symbol RC, as described in section 4.

acn — Application connection number assigned by the network software to the program end of the logical connection on which the break occurred. This field always contains a nonzero value previously used by the application program in an FC/INIT/N message and must be used by the application program in a subsequent FC/RST/R message before data transmission on the connection is again possible. This field can be accessed with the reserved symbol FCACN, as described in section 4.

abn — Application block number assigned by the application program to the last block for which an FC/ACK/R message was generated before the break condition occurred. This field contains a zero only when no FC/ACK/R messages were generated before the break occurred. This field can be accessed with the reserved symbol FCABN, as described in section 4.

reserved — Reserved for CDC. Reserved fields must be equal to zero.

Figure 3-27. Break (FC/BRK/R) Supervisory Message Format



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

fc — Primary function code 83_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol FC.

rst — Secondary function code 1. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol RST.

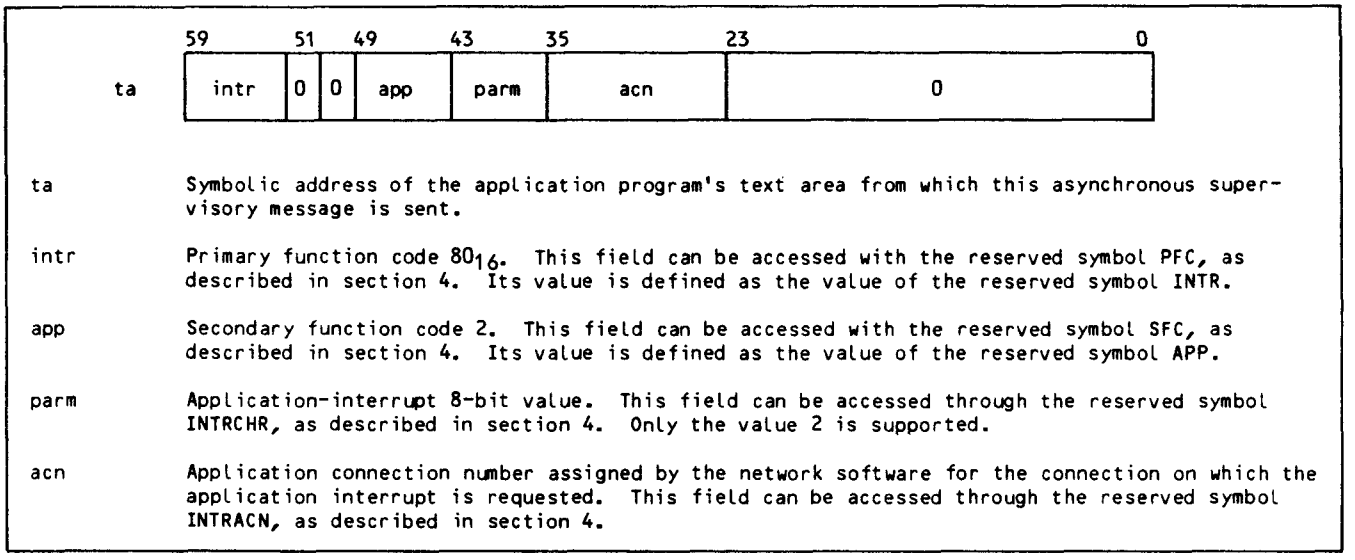
acn — Application connection number assigned by the network software to the program end of the logical connection to be reset. This value is always nonzero and must be the acn value received by the application program in a previous FC/BRK/R message. This field can be accessed with the reserved symbol FCACN, as described in section 4.

Figure 3-28. Reset (FC/RST/R) Supervisory Message Format

The break message reflects suspension of downline traffic only. Upline traffic (input) on the connection is not affected. The application block number from the last block-delivered message is the only reliable guide to the downline blocks discarded and requiring retransmission.

The asynchronous application-interrupt supervisory message (figure 3-29) provides a method for the application program to bypass data and to communicate with the other end of the connection.

For application-to-terminal connections, this allows an application program to send an 8-bit ASCII character as expedited data to the network software. The meaning of this character is predefined by the network software. The only currently predefined use is for asynchronous device connections with the 8-bit ASCII parameter set to 2, meaning that the application wants downline data discarded. NAM responds to the application-interrupt supervisory message by sending the application-interrupt response message shown in figure 3-30. The application cannot send another INTR/APP until the response is received. The network software discards all downline data queued for the terminal until a terminate-output-marker synchronous supervisory message (figure 3-31) is received. The TO/MARK/R message serves as a marker to indicate the point at which network software stops discarding downline data blocks queued for the terminal after it receives the application originated interrupt. It is the responsibility of the application to send the TO/MARK/R supervisory message on the connection.



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

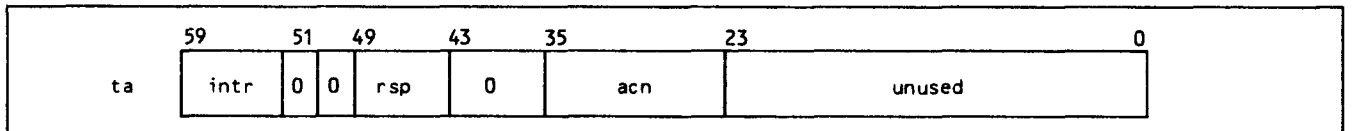
intr — Primary function code 80_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol INTR.

app — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol APP.

parm — Application-interrupt 8-bit value. This field can be accessed through the reserved symbol INTRCHR, as described in section 4. Only the value 2 is supported.

acn — Application connection number assigned by the network software for the connection on which the application interrupt is requested. This field can be accessed through the reserved symbol INTRACN, as described in section 4.

Figure 3-29. Application-Interrupt (INTR/APP/R) Supervisory Message Format



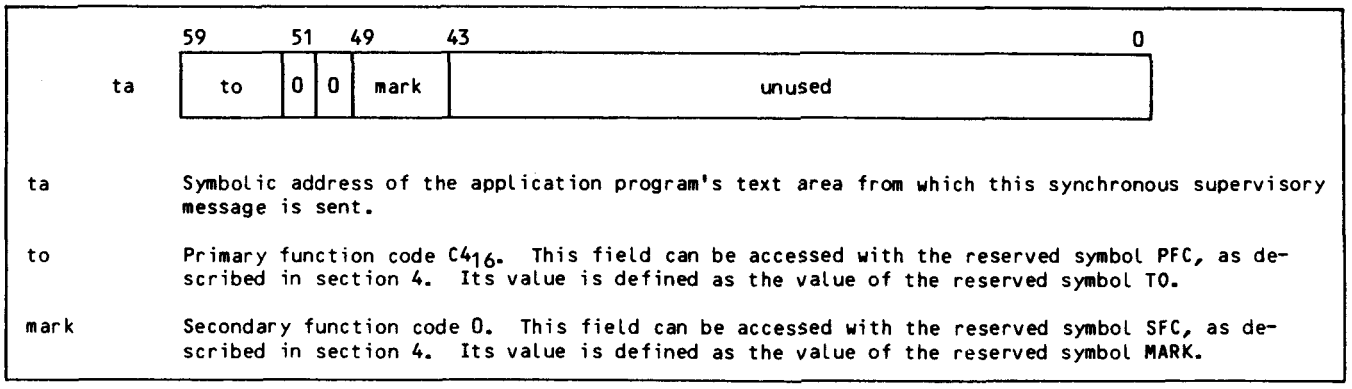
ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

intr — Primary function code 80_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol INTR.

rsp — Secondary function code 01. This field can be accessed with the reserved symbol SFC, as defined in section 4. Its value is defined as the value of the reserved symbol RSP.

acn — Application connection number assigned by the network software for the connection on which the user-interrupt-response supervisory message was sent. The value placed in this field must be the device connection value used in the INTR/USR/R message to which this message is a response. This field can be accessed with the reserved symbol INTRACN, as described in section 4.

Figure 3-30. Application-Interrupt-Response (INTR/RSP/R) Supervisory Message Format



ta — Symbolic address of the application program's text area from which this synchronous supervisory message is sent.

to — Primary function code $C4_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol TO.

mark — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol MARK.

Figure 3-31. Terminate-Output-Marker (TO/MARK/R) Supervisory Message Format

USING USER-INTERRUPT FEATURE

The terminal operator can send a message to the application that bypasses regular upline data by entering a user-interrupt sequence. The operator enters the interrupt sequence by entering the TIP command control character (defined by the CT command) and an alphabetic character followed by an end-of-block indicator. NAM generates the user-interrupt-request supervisory message, INTR/USR/R (illustrated in figure 3-32) and sends it to the application.

The application program responds with the application-interrupt-response supervisory message (illustrated in figure 3-30) after receiving the INTR/USR/R message if the application supports user interrupts. If the application does not support user interrupts, it can ignore the INTR/USR/R message and issues no response. Figure 3-33 illustrates the flow of messages. Until the response is sent, the user cannot enter another interrupt sequence.

If the application program supports user interrupts, predefined meanings can be given to the alphabetic characters available as interrupt characters.

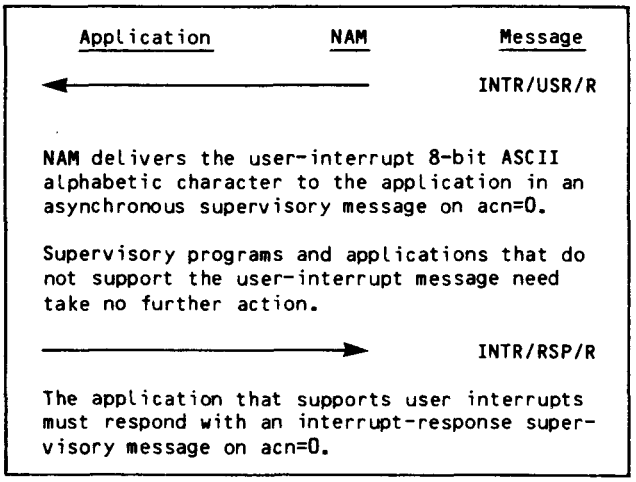
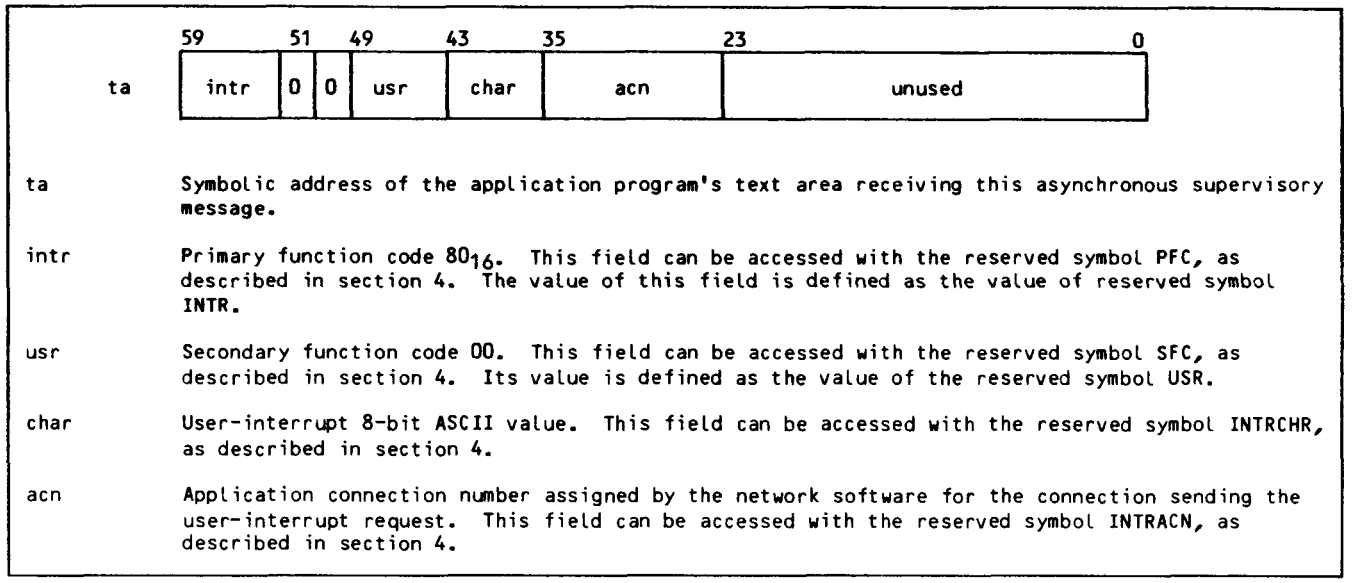


Figure 3-33. User-Interrupt Message Sequence



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

intr — Primary function code 80_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. The value of this field is defined as the value of reserved symbol INTR.

usr — Secondary function code 00. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol USR.

char — User-interrupt 8-bit ASCII value. This field can be accessed with the reserved symbol INTRCHR, as described in section 4.

acn — Application connection number assigned by the network software for the connection sending the user-interrupt request. This field can be accessed with the reserved symbol INTRACN, as described in section 4.

Figure 3-32. User-Interrupt-Request (INTR/USR/R) Supervisory Message Format

CONVERTING DATA

Data exchanged on an interactive terminal-to-application connection is converted to and from display code or ASCII character codes at the discretion of the application program. This conversion also includes packing and unpacking of data character codes from bytes of different sizes. NAM converts data in a given block according to the application character type associated with the block.

Data sent downline by an application program for output at an interactive terminal or to another application has an application character type associated with it on a block-by-block basis. When the application program needs to change the conversion performed for downline data on a given connection, it simply changes the act field value used in the block header of each data block. The effects of a given act field value declaration are described in detail in section 2.

Data received upline by an application program from a console device or another application has an application character type associated with the logical connection on which the data blocks are received. The application character type associated with a given block of upline (input) data is assigned by the application program when the logical connection is first established. This assignment is part of the connection-accepted supervisory message. When the application program needs to change the conversion performed for upline data on a given connection, it changes the act field value associated with the logical connection by issuing the asynchronous change-input-character-type supervisory message. This message can be issued at any time the logical connection exists, after the application program has issued the FC/INIT/N message for the connection. As shown in figure 3-34, there is no response to the change-input-character-type message, but the message takes effect immediately.

Application	NAM	Message
	⟶	DC/CICT/R

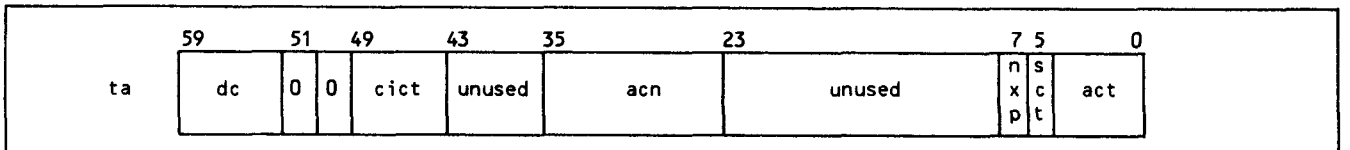
The next input request for this logical connection returns blocks in bytes of the new character type.

Figure 3-34. Change Input Character Type Message Sequence

The change-input-character-type message has the format shown in figure 3-35. The act field values described in the figure are explained in more detail in section 2. Note that transparent mode upline data cannot be correctly received when an application character type other than 2 or 3 is associated with the logical connection.

The conversion change requested by the change-input-character-type message affects the next block fetched by the application program. For example, the application program might have been receiving blocks of 7-bit ASCII code characters, packed in 12-bit bytes (an act value of 3); the application program now needs to receive blocks of 6-bit display code characters, packed in 6-bit bytes (an act value of 4). The program sends a change-input-character-type message, specifying an act value of 4; the next block received from that logical connection is 6-bit display code characters, packed in 6-bit bytes.

An application program can also change the format in which it will receive synchronous supervisory messages from character type 2 to 3 or vice versa. The third parameter the user can change with the change-input-character-type supervisory message is the no-transparent-input-flag. The initial values are specified on the connection-accepted supervisory message.



ta — Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.

dc — Primary function code $C2_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol DC.

cict — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CICT.

acn — Application connection number assigned by the network software to this end of the logical connection when it was established. The value placed in this field must be the value associated with an existing connection and used in the FC/INIT/N supervisory message that completed initialization of the connection. This field can be accessed with the reserved symbol DCACN, as described in section 4.

Figure 3-35. Change-Input-Character-Type (DC/CICT/R) Supervisory Message Format (Sheet 1 of 2)

nxp — No-transparent-input flag. This field can have the values:

- 0 — Allows transparent input
- 1 — No transparent input allowed

This field can be accessed with the reserved symbol DCNXP, as described in section 4.

sct — Application character type in which the application program expects to receive synchronous supervisory messages. This field can have the values:

- 0 — Deliver supervisory messages in character type 2
- 1 — Deliver supervisory messages in character type 3

This field can be accessed with the reserved symbol DCSCT, as described in section 4.

act — Application character type, specifying the form of character byte packing that the application program requires for all future input data blocks from the logical connection. The value declared replaces the value previously declared by the application program for this connection in a CON/REQ/N or DC/CICT/R message. This field can have the values:

- 1 — 60-bit words; must be used for supervisory messages with ADR=0; can be used for intra-host application-to-application connections (for restrictions see the description of the IBU bit in section 2); cannot be used for terminal-to-application connections or for inter-host application-to-application connections.
- 2 — 8-bit characters in 8-bit bytes, packed 7.5 characters per central memory word; if the input is not transparent mode, the ASCII character set described in table A-2 is used.
- 3 — 8-bit characters in 12-bit bytes, packed 5 characters per central memory word, right-justified with zero fill within each byte; if the input is not transparent mode, the ASCII character set described in table A-2 is used.
- 4 — 6-bit display code characters in 6-bit bytes, packed 10 characters per central memory word; the characters used are the ASCII set of CDC characters described in table A-1. This applies to terminal-to-application connections only.
- 5 thru 11 — Reserved for CDC.
- 12 thru 15 — Reserved for installation.

The act value declared applies only to input on the connection and can be changed by another DC/CICT/R message at any time during the existence of this logical connection. This field can be accessed with the reserved symbol CONACT, as described in section 4.

Figure 3-35. Change-Input-Character-Type (DC/CICT/R) Supervisory Message Format (Sheet 2 of 2)

If the requested application character type is not valid for the connection specified, a logical-error supervisory message is sent to the application program, and the application character type associated with the logical connection is unchanged. Otherwise, receipt of the change-input-character-type message is not acknowledged.

TRUNCATING DATA

Data received upline by an application program from a terminal or from another application can be truncated to fit the text area buffer provided by the application. This truncation allows the application to obtain at least part of a block longer

than the text area instead of receiving an input-block-undeliverable reply (ibu bit set in the block header). An asynchronous supervisory message is available to inform NAM that the application wants to have a block truncated on a particular connection or to have blocks truncated on all existing and future connections. As indicated in figure 3-36, the effect of this supervisory message is irreversible, and there is no response.

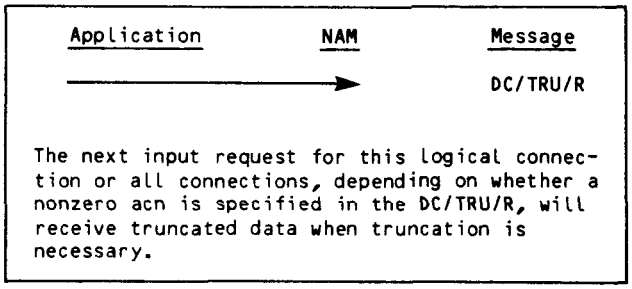


Figure 3-36. Data Truncation Message Sequence

When data is actually truncated, the tru bit in the application block header is set, and the tlc field in the block header is set to the size of the portion of the block received (instead of being set to the full size of the block).

This data truncation supervisory message (figure 3-37) can be issued at any time after completion of a NETON call. This message affects all messages on the connection, including synchronous supervisory messages. If acn=0 is specified, the application has to call NETOFF and NETON again to not receive truncated data blocks. If the acn field specified within the message identifies a nonexistent logical connection, a logical-error supervisory message is sent to the application and data truncation does not occur. If more than one data truncation message affecting a connection is issued, the extra messages are ignored.

CHANGING TERMINAL CHARACTERISTICS

The process of configuring a terminal consists of defining a number of terminal characteristics that the network software should use in communication with a terminal. Some terminal characteristics can be given default values by the Communications Control Program (CCP), while others can be provided by the Network Definition Language (NDL) and the site administrator.

Once a terminal is configured (or defined), subsequent changes to the terminal definition can be made via TIP commands by the terminal operator, or via supervisory messages by the application program to which the terminal is connected.

This subsection describes the supervisory messages that the application can use to change the settings of terminal characteristics. The supervisory message used to find out the current values of terminal characteristics is described in the following subsection, Requesting Terminal Characteristics. Terminal definition commands are described in appendix F.

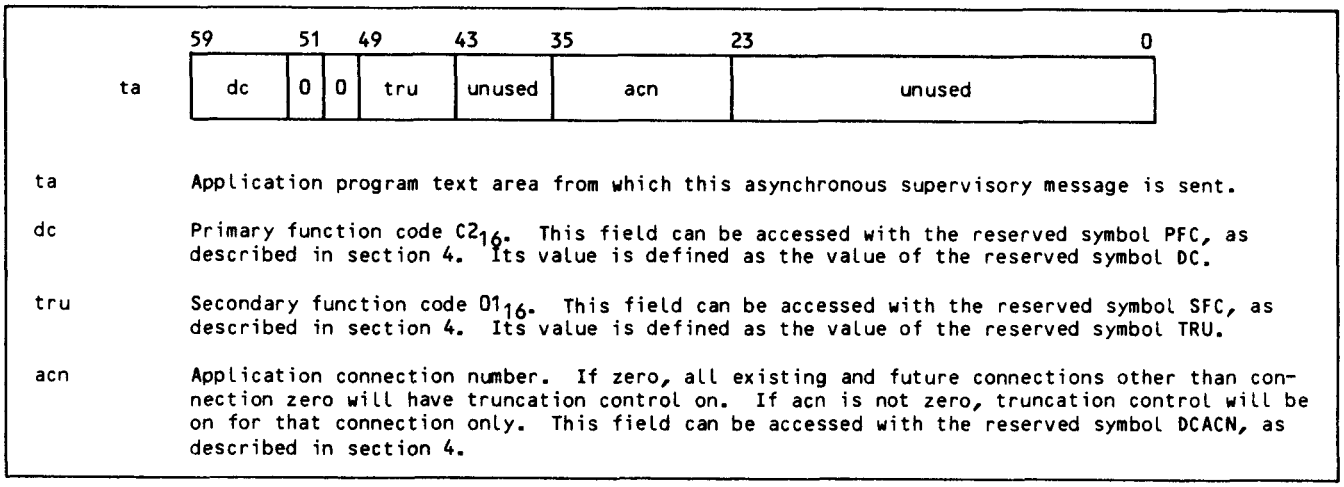
Figure 3-38 shows some possible message sequences involved in changing terminal characteristics.

The application program is advised of the TIP command entry explicitly only when the command changes one of three terminal characteristics:

Terminal class (value describing the physical attributes of a group of similar terminals)

Page width (value describing the number of characters potentially output per line)

Page length (value describing the number of potential lines output per page)



ta — Application program text area from which this asynchronous supervisory message is sent.

dc — Primary function code $C2_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol DC.

tru — Secondary function code 01_{16}. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol TRU.

acn — Application connection number. If zero, all existing and future connections other than connection zero will have truncation control on. If acn is not zero, truncation control will be on for that connection only. This field can be accessed with the reserved symbol DCACN, as described in section 4.

Figure 3-37. Data Truncation (DC/TRU/R) Supervisory Message Format

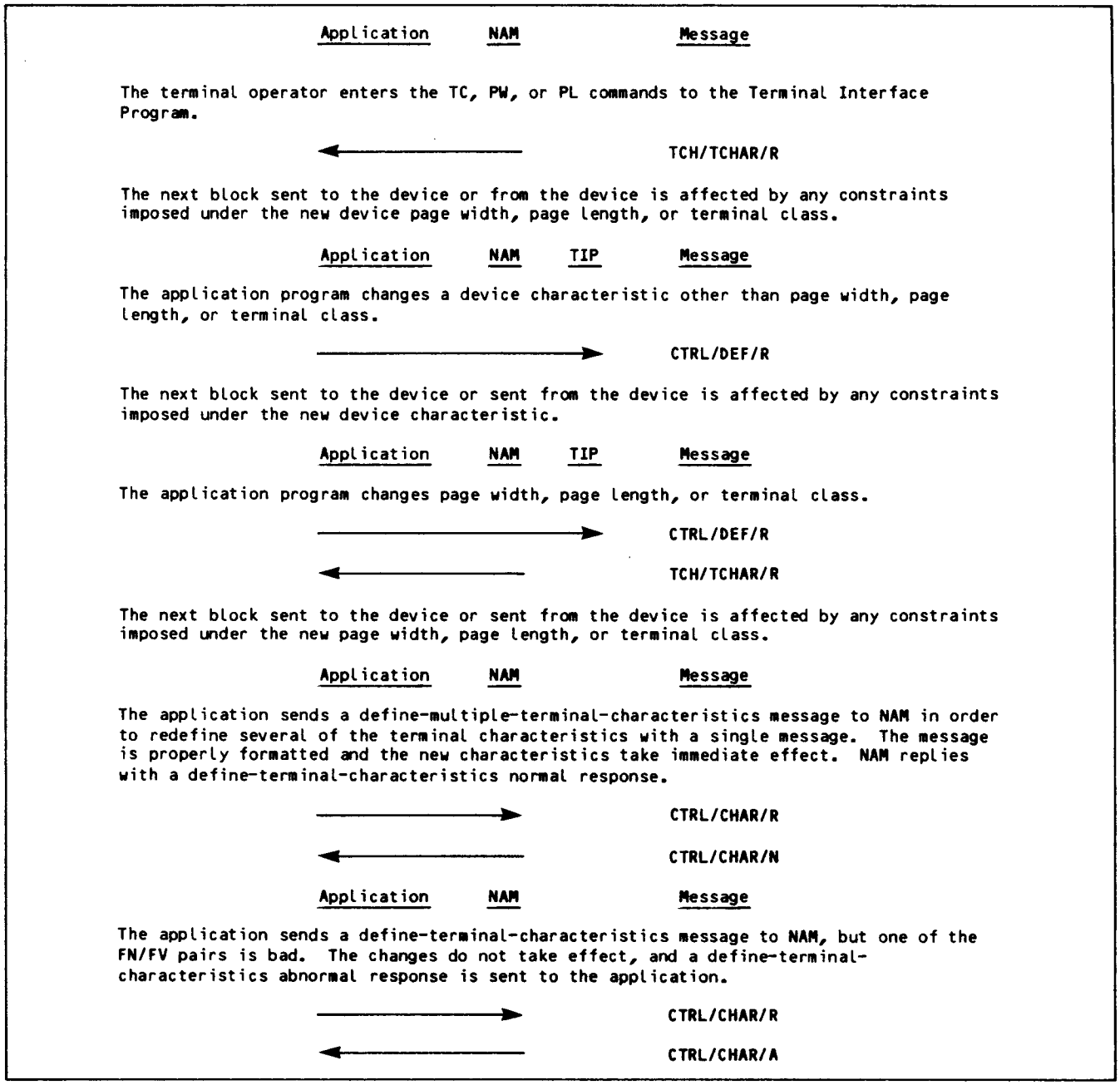
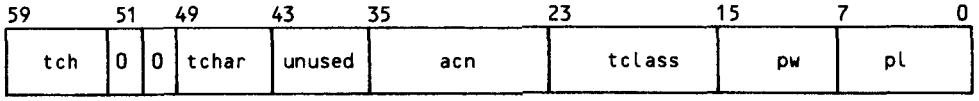


Figure 3-38. Terminal Characteristics Redefinition Message Sequences

The upline terminal-characteristics-redefined supervisory message is an asynchronous one, with the format shown in figure 3-39. This message is sent to the application by NAM whenever NAM is notified that one of the above terminal characteristics has been redefined by a terminal user or an application program. The effect of the TIP command causing this message is immediate, and no response is required from the application program.

The application is provided with two different formats for changing terminal characteristics. The define-terminal-characteristics supervisory message (figure 3-40) specifies terminal characteristic commands as a string of ASCII characters. If there is an error in one of the commands, the TIP stops processing the message, no indication is sent to the application, and any commands prior to the error are processed. There is no response to this message.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

tch — Primary function code 64_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol TCH.

tchar — Secondary function code 0. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol TCHAR.

acn — Application connection number assigned by the network software to this end of the logical connection for which the change occurred. This field always contains a value previously used by the application program in an FC/INIT/N message. This field can be accessed with the reserved symbol CONACN, as described in section 4.

tclass — The terminal class currently associated with the real terminal by the TIP servicing it. The terminal class determines the parameters and ranges valid for redefinition of the terminal. The terminal is serviced by the TIP according to the attributes associated with the terminal class (see text). The terminal class field can contain the values:

1 Archetype terminal for the class is a Teletype Corporation Model 30 Series.

2 Archetype terminal for the class is a CDC 713-10, 751-1, 756.

3 Reserved for CDC use (cannot be received in current version of software).

4 Archetype terminal for the class is an IBM 2741.

5 Archetype terminal for the class is a Teletype Corporation Model 40-2.

6 Archetype terminal for the class is a Hazeltine 2000, operating as a teletypewriter.

7 Archetype terminal for the class is a CDC 752.

8 Archetype terminal for the class is a Tektronix 4000 Series, operating as a teletypewriter.

9 Archetype terminal for the class is a HASP (post-print) protocol multileaving workstation.

10 Archetype terminal for the class is a CDC 200 User Terminal.

11 Archetype terminal for the class is a CDC 714-30.

12 Archetype terminal for the class is a CDC 711-10.

13 Archetype terminal for the class is a CDC 714-10/20.

14 Archetype terminal for the class is a HASP (pre-print) protocol multileaving workstation.

15 Archetype terminal for the class is a CDC 734.

16 Archetype terminal for the class is an IBM 2780.

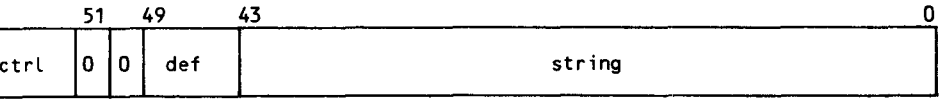
17 Archetype terminal for the class is an IBM 3780.

If the terminal class value received has not changed from that previously associated with the terminal, then the value in either the pw or pl fields (or both) has usually changed. If the terminal class value received has changed from that previously associated with the terminal, then all attributes associated with the terminal have been changed to the default attributes for the new terminal class; the values in the pw and pl fields might have changed from those previously associated with the real device. This field can be accessed by the reserved symbol TCHTCL, as described in section 4.

pw — The most recently declared page width of the terminal, specifying the number of characters in a physical line of output. This field can contain the values 0 or $20 \leq pw \leq 255$. This field can be accessed by the reserved symbol TCHPW, as described in section 4.

pl — The most recently declared page length of the terminal, specifying the number of physical lines that constitute a page. This field can contain the values 0 or $8 \leq pl \leq 255$. This field can be accessed by the reserved symbol TCHPL, as described in section 4.

Figure 3-39. Terminal-Characteristics-Redefined (TCH/TCHAR/R) Supervisory Message Format



ta — Symbolic address of the application program's text area from which this synchronous supervisory message is sent.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the reserved symbol CTRL.

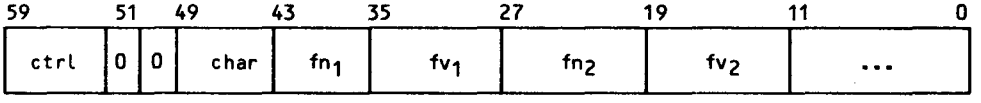
def — Secondary function code 4. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol DEFF.

string — A 7-bit ASCII character string that describes one or more commands consisting of the characteristic and its desired setting. The characteristic and its value are separated by an equals sign. Multiple characteristics can be changed by separating the commands by the network control character. See appendix F for a list of all the possible commands that can be sent.

Figure 3-40. Define-Terminal-Characteristics (CTRL/DEF/R) Supervisory Message Format

The define-multiple-terminal-characteristics message is described in figure 3-41. This message specifies a string of pairs of 8-bit numbers starting after the secondary function code field and extending for as many (8-bit) bytes as necessary. The application stores an 8-bit field number (FN) in the first of a pair of bytes and a field value (FV) in the second byte of the pair. Each FN represents a particular device characteristic, and the corresponding FV represents the value the application program wishes to assign to that characteristic. The application program needs to specify only the FN/FV pairs for the characteristic it wants to change. If one of the FN/FV pairs contains an incorrect value, no characteristics are changed and the application program receives the abnormal response message shown in figure 3-42. Figure 3-43 shows the normal response to the define-multiple-terminal-characteristics supervisory message. Valid combinations of FN/FV pairs are defined in table 3-2.

The define-terminal-characteristics and define-multiple-terminal characteristics supervisory messages sent downline by the application program are removed from the output stream by the TIP and acted on directly. The terminal operator is not advised of their occurrence in the output stream.



ta — Symbolic address of the application program text area from which this synchronous supervisory message is sent.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

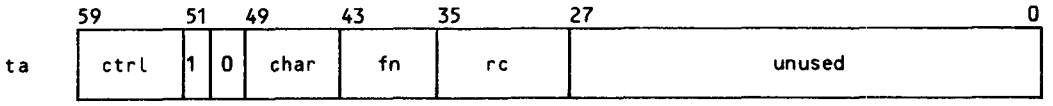
char — Secondary function code 8. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CHAR.

fn_i — The 8-bit field number of the desired parameter.

fv_i — The 8-bit field value of the desired parameter.

The field number and field value pairs can extend beyond one word. Valid field numbers and values are defined in table 3-2.

Figure 3-41. Define-Multiple-Terminal-Characteristics (CTRL/CHAR/R) Supervisory Message Format



ta — Symbolic address of the application program text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

char — Secondary function code 8. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CHAR.

fn — Field number causing the abnormal response.

rc — Reason code for error. This field can have the values:

1 Out of range value

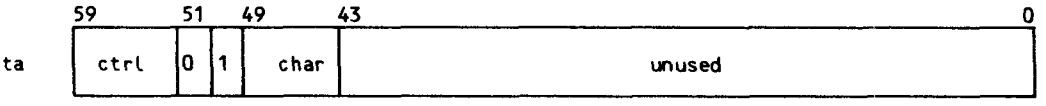
2 Duplicate control character

3 Invalid value for terminal class

4 Illegal terminal class change

5 Illegal parameter for terminal class

Figure 3-42. Define-Multiple-Terminal-Characteristics Abnormal Response (CTRL/CHAR/A) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

char — Secondary function code 8. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CHAR.

Figure 3-43. Multiple-Terminal-Characteristics-Defined (CTRL/CHAR/N) Supervisory Message Format

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES

Parameter (Mnemonic)	Number (Octal)	Terminal Classes (1)	Value Range (Decimal)	Form of Input
Abort block (AB)	51	1-8 (9-17) (2)	0-127	Numerical value for character (3)
Break as user break 1 (BR)	63	1, 2, 5-8 (4, 9-17)	0-1	Yes (1), no (0)
Backspace character (BS)	47	1-8 (9-17)	0-127	Numerical value for character (3)
Interruption character (B1)	52	1-15 (16, 17)	0-127	Numerical value for character (3)

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES (Contd)

Parameter (Mnemonic)	Number (Octal)	Terminal Classes (1)	Value Range (Decimal)	Form of Input
Termination character (B2)	53	1-15 (16, 17)	0-127	Numerical value for character (3)
Carriage return idle count (CI)	54	1-8 (9-17)	0-99	Decimal number
	56	1-8 (9-17)	0-1	CA (1)
Cancel character (CN)	46	1-15 (16, 17)	0-127	Numerical value for character (3)
Cursor positioning (CP)	107	1, 2, 5-8 (4, 9-17)	0-1	Yes (1), no (0)
Network control character (CT)	50	1-17	0-127	Numerical value for character (3)
Single message transparent input delimiters (DL) (4)	70	1-8 (9-17)	0-1	Character specified (1), not specified (0)
	71	1, 2, 5-8 (9-17)	0-15	Character count (upper byte)
	72	1, 2, 5-8 (9-17)	0-255	Character count (upper byte)
	73	1-8, 10-13, 15 (9, 14, 16, 17)	0-255 (5)	Numerical value for character (Xhh)
	74	1, 2, 5-8 (9-17)	0-1	Timeout (1), no timeout (0)
	106	1-8, 10-13, 15	0	Single message (0)
End-of-block character (EB)	100	1, 2, 5-8, 10-13, 15	0-127 (5)	Numerical value for character
	101	1, 2, 5-8, 10-13, 15	1-2 (5)	EL (1), EB (2)
	102	1, 2, 5-8, 10-13, 15 (9, 14, 16, 17)	0-3 (5)	NO (0), CR (1), LF (2), CL (3)
End-of-line character (EL)	75	1, 2, 5-8, 10-13, 15	0-127 (5)	Numerical value for character
	76	1, 2, 5-8, 10-13, 15	1-2	EL (1), EB (2)
	77	1, 2, 5-8, 10-13, 15 (9, 14, 16, 17)	0-3 (5)	NO (0), CR (1), LF (2), CL (3)
Echoplex mode (EP)	61	1, 2, 5-8 (4, 9-17) (2)	0-1	Yes (1), no (0)
Full ASCII input (FA)	67	1-8, 10-13, 15	0-1	Yes (1), no (0)
Host availability display (HD)	41	1-17	0-1	Yes (1), no (0)
Input control (IC)	103	1, 2, 5-8 (4, 9-17) (2)	0-1	Yes (1), no (0)

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES (Contd)

Parameter (Mnemonic)	Number (Octal)	Terminal Classes (1)	Value Range (Decimal)	Form of Input
Input device (IN)	64	1-8, 10-13, 15 (6)	0-1	Transparent input (1), not transparent (0)
	65	1-8 (6)	0-2 (5)	KB (0), PT (1), BK (2)
Line feed idle count (LI)	55	1-8 (9-17)	0-99	Decimal number
	57	1-8 (9-17)	0-1	CA (1)
Lockout unsolicited messages (LK)	40	1-15 (16, 17)	0-1	Yes (1), no (0)
Output control (OC)	104	1, 2, 5-8, (4, 9-17) (2)	0-1	Yes (1), no (0)
Output device (OP)	66	1-8 (9-17)	0-2 (5)	DI (0), PR (1), PT (2)
Parity processing (PA)	62	1, 2, 5-8	0-3	Z (0), O (1), E (2), N (3)
Page waiting (PG)	45	1-8, 10-13, 15 (9, 14, 16, 17)	0-1	Yes (1), no (0)
Page length (PL)	44	1-17	0, 8-255	Decimal number
Page width (PW)	43	1-17	0, 20-255	Decimal number
Special editing mode (SE)	60	1-8 (9-17) (6)	0-1	Yes (1), no (0)
Terminal class (TC)	42	1-17	1-17 (5)	Decimal number
Multi-message transparent input delimiter (XL) (4)	70	1-8 (9-17)	0-1	Character specified (1), not specified (0)
	71	1, 2, 5-8 (9-17)	0-15	Character count (upper byte)
	72	1, 2, 5-8 (9-17)	0-255	Character count (lower byte)
	73	1-8, 10-13, 15 (9, 14, 16, 17)	0-255 (5)	Numerical value for character (Xhh)
	74	1, 2, 5-8 (9-17)	0-1	Timeout (1), no timeout (0)
	105	1-8 (9-17)	0-255 (5)	Numerical value for character (Yhh)
	106	1-8, 10-13, 15	1	Multi-message (1)

Notes:

(1) Numbers in parentheses in this column indicate terminal classes for which the parameter is ignored.

(2) Ignored for packet-switching network (PSN) terminals.

(3) Any hexadecimal value except 00-02, 20, 30-39, 3D, 41-5A, 61-7A, or 7F.

(4) If the value of one of the fields for this parameter is changed, the values of all other fields for this parameter must also be specified.

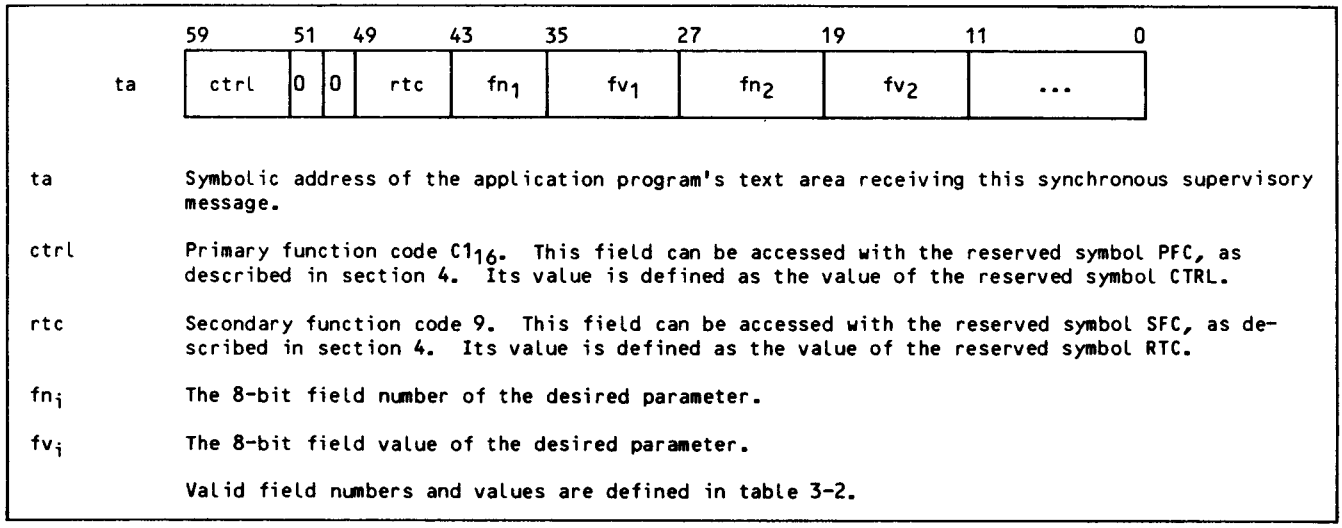
(5) Not all values are legal for all terminal classes.

(6) Not allowed for packet-switching network (PSN) terminals.

REQUESTING TERMINAL CHARACTERISTICS

The request-terminal-characteristics supervisory message (figure 3-44) is issued by an application program on console connections to learn the current value of the terminal characteristics. The application program specifies a string of pairs of 8-bit numbers starting after the secondary function code field and extending for as many 8-bit bytes as necessary. The application stores a field number (FN) in the first half (8 bits) of the 8-bit pair and reserves the second half (8 bits) for a field value (FV). Each FN represents a particular characteristic. The network returns the value of the characteristic in the corresponding FV byte. Any value placed in the FV byte by the application is ignored and overwritten. The application program needs to specify only the FNs for the characteristics it is interested in. If the string contains an incorrect FN, no terminal characteristics are returned and the application receives the abnormal response message shown in figure 3-45. For a list of legal FNs and the corresponding range of possible FVs see table 3-2.

The response to a request-terminal-characteristics supervisory message is a terminal-characteristics-definition message (figure 3-46). This message can be received only on console connections. The NPU generates a string of pairs of 8-bit numbers starting after the secondary function code field and extending for as many 8-bit bytes as necessary. The first 8-bits of the 16-bit pair is one of the field numbers specified in the request-terminal-characteristics supervisory message. The second 8-bits of the 16-bit pair is the current value of the particular characteristic the FN represents. For a list of valid FNs and the associated valid range of FVs see table 3-2.



ta — Symbolic address of the application program's text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

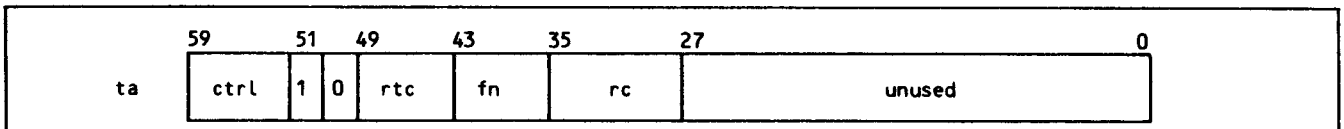
rtc — Secondary function code 9. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol RTC.

fn_i — The 8-bit field number of the desired parameter.

fv_i — The 8-bit field value of the desired parameter.

Valid field numbers and values are defined in table 3-2.

Figure 3-44. Request-Terminal-Characteristics (CTRL/RTC/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

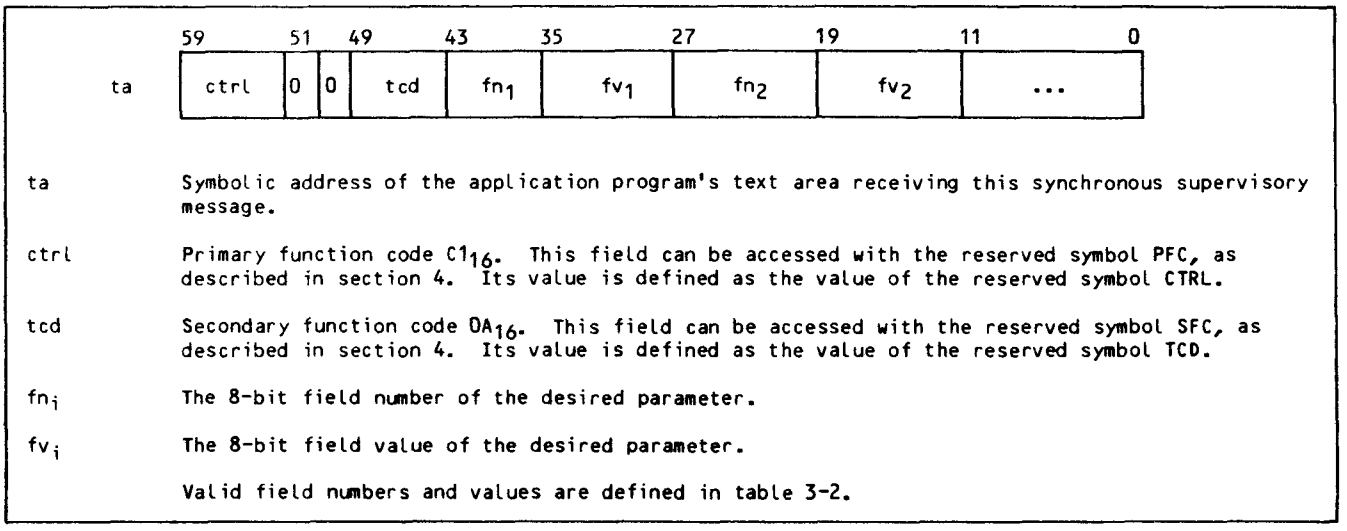
rtc — Secondary function code 9. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol RTC.

fn — First field number in the string found to be erroneous by the network software. In case of several bad field numbers, only the first bad one will be diagnosed.

rc — Reason code for error. This field can have the value:

5 Illegal fn value

Figure 3-45. Request-Terminal-Characteristics Abnormal Response (CTRL/RTC/A) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol CTRL.

tcd — Secondary function code $0A_{16}$. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol TCD.

fn_i — The 8-bit field number of the desired parameter.

fv_i — The 8-bit field value of the desired parameter.

Valid field numbers and values are defined in table 3-2.

Figure 3-46. Terminal-Characteristics-Definition (CTRL/TCD/R) Supervisory Message Format

HOST OPERATOR COMMUNICATION

The host operator supervisory messages described in this subsection are not used in this release. However, they will be used in future releases and applications will have to be able to either handle or ignore them.

The host operator request-to-activate-debug-code supervisory message (figure 3-47) is sent from NAM to the application program when the operator enters the K-display command:

K.DB=appname

The application should turn on the in-line debug code. Activating the in-line debug code can change the application program's abort conditions or error case handling or both. There is no response to the request-to-activate-debug-code message.

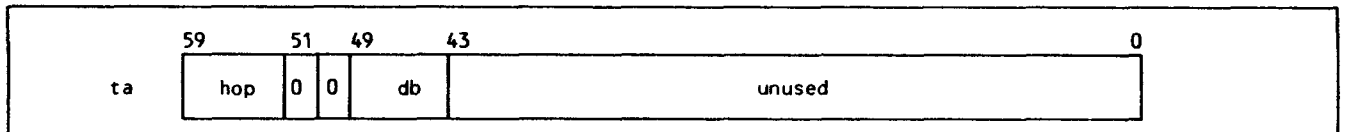
The host operator request-to-turn-off-debug-code supervisory message shown in figure 3-48 is sent from NAM to the application program when the operator enters the K-display command:

K.DE=appname

The application should turn off its in-line debug code. There is no response to the request-to-turn-off-debug-code message.

The host operator request-to-dump-field-length supervisory message (figure 3-49) is sent from NAM to the application program when the operator enters the K-display command:

K.DU=appname

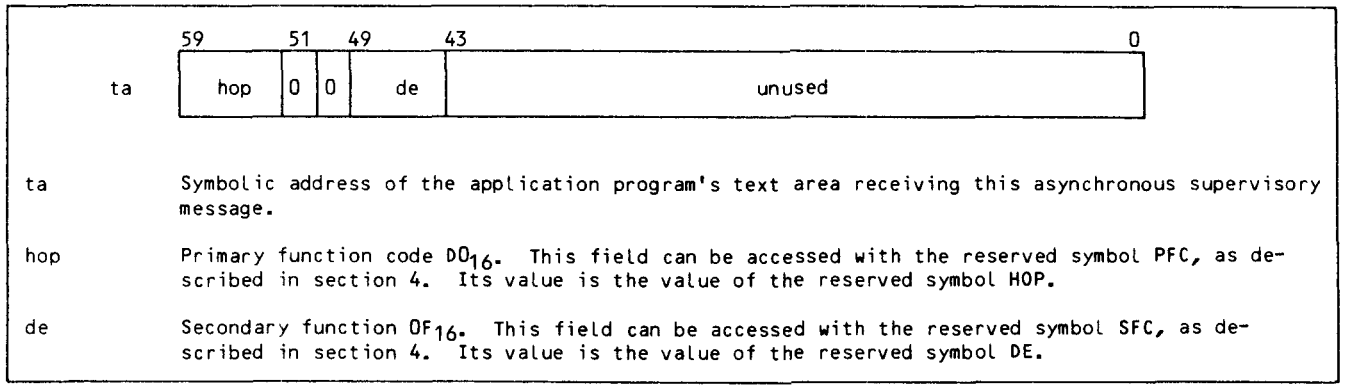


ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

db — Secondary function code $0E_{16}$. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DB.

Figure 3-47. Host Operator Request-to-Activate-Debug-Code (HOP/DB/R) Supervisory Message Format

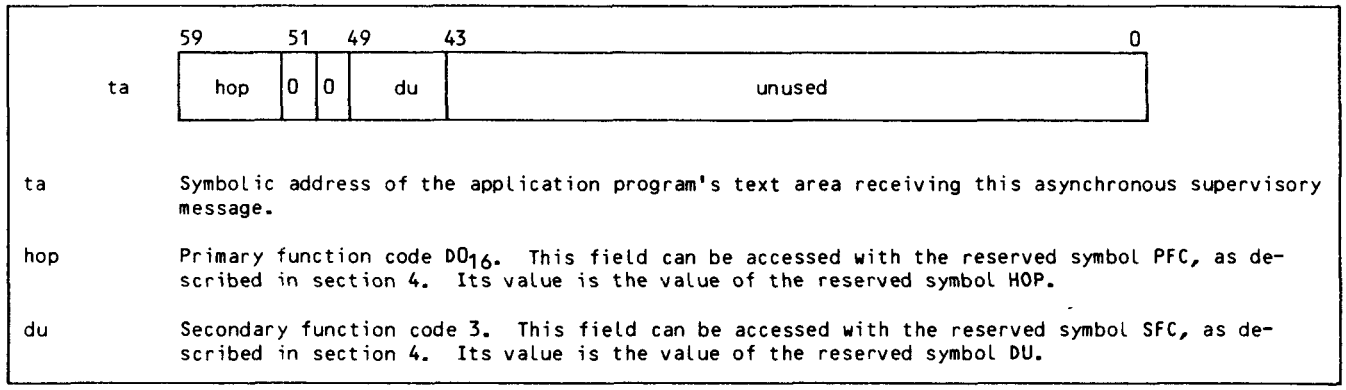


ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

de — Secondary function $0F_{16}$. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DE.

Figure 3-48. Host Operator Request-to-Turn-Off-Debug-Code (HOP/DE/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

du — Secondary function code 3. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DU.

Figure 3-49. Host Operator Request-to-Dump-Field-Length (HOP/DU/R) Supervisory Message Format

The application should dump its field length. The application can call NETDMB to dump its field length onto the AIP dump file ZZZZDMB (see section 6). There is no response to the request-to-dump-field-length message.

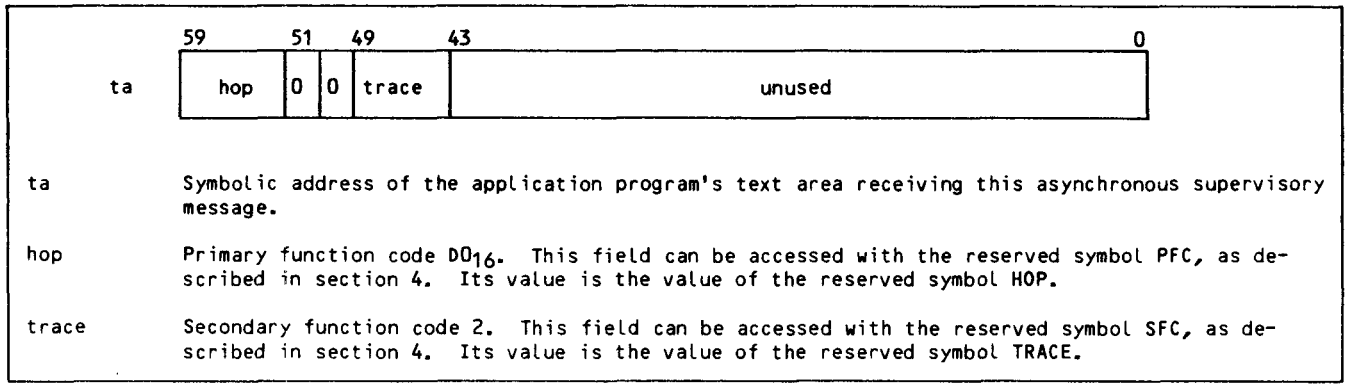
The host operator request-to-turn-AIP-tracing-on supervisory message (figure 3-50) is sent from NAM to the application program when the operator enters the K-display command:

K.LB=appname

The application program should begin logging of network traffic on the debug log file. The application program should call NETDBG to turn AIP tracing on. Note that the application program must be loaded with NETIOD for the AIP tracing to occur. There is no response to the request-to-turn-AIP-tracing-on message.

The host operator request-to-turn-AIP-tracing-off supervisory message (figure 3-51) is sent from NAM to the application program when the operator enters the K-display command:

K.LE=appname

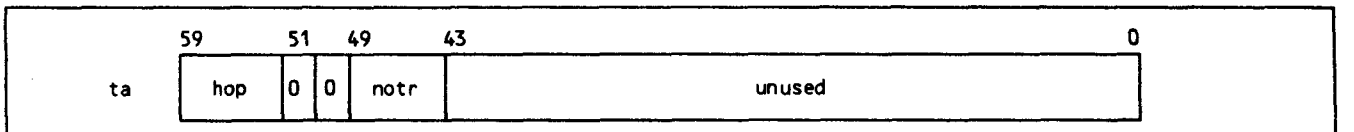


ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

trace — Secondary function code 2. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol TRACE.

Figure 3-50. Host Operator Request-to-Turn-AIP-Tracing-On (HOP/TRACE/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

notr — Secondary function code 7. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol NOTR.

Figure 3-51. Host Operator Request-to-Turn-AIP-Tracing-Off (HOP/NOTR/R) Supervisory Message Format

The application program should stop logging network traffic in its debug log file. The application program should call NETDBG to turn AIP tracing off. There is no response to the request-to-turn-AIP-tracing-off supervisory message.

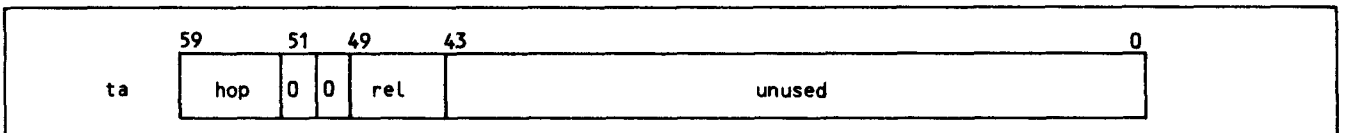
The host operator request-to-release-debug-log-file supervisory message (figure 3-52) is sent from NAM to the application program when the operator enters the K-display command:

K.LR=appname

The application program should release its debug log file. The application program should call NETREL to release the debug log file. To ensure proper processing of the debug log file, the application program must have issued a prior NETREL call as described in section 6. There is no response to the request-to-release-debug-log-file supervisory message.

The host operator request-to-restart-statistics-gathering supervisory message (figure 3-53) is sent from NAM to the application program when the operator enters the K-display command:

K.RS=appname

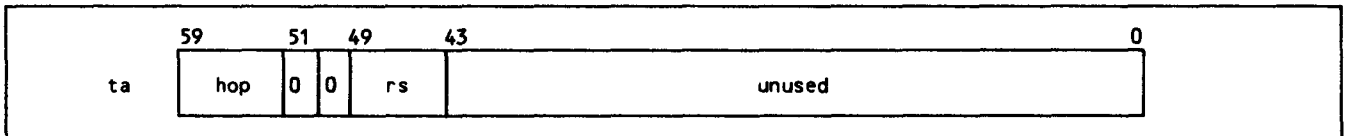


ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

rel — Secondary function code $0D_{16}$. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol REL.

Figure 3-52. Host Operator Request-to-Release-Debug-Log-File (HOP/REL/R) Supervisory Message Format



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

hop — Primary function code $D0_{16}$. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

rs — Secondary function code 8. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol RS.

Figure 3-53. Host Operator Request-to-Restart-Statistics-Gathering (HOP/RS/R) Supervisory Message Format

The application program should flush its statistics counters, reset them to zero, and restart statistics gathering. For this supervisory message to be useful the application program should do at least one of the following:

Restart AIP statistics gathering by calling NETSTC (described in section 6) to turn AIP statistics gathering off or back on

Restart any other statistical information internal to the application program that can be used to tune the particular application. The application program can write such statistical information onto the AIP statistical file ZZZZZSN by calling NETLGS (see section 6).

There is no response to the request-to-restart-statistics-gathering message.

HOST SHUTDOWN

Conditions sometimes require the host operator to terminate network operations or to abort the application program. The host operator can shut down the entire data communications network or portions of the network, element by element, including executing application programs.

The operator has two shutdown options available. He can select an idle-down option that permits gradual termination of operations, usually as a normal part of network service. He can also select a disable option; this option requests immediate termination of application program operations and can either follow selection of the idle-down option or be independently selected.

The type of shutdown determines the shutdown processing that should be performed by the application program. Figure 3-54 illustrates the three asynchronous supervisory message sequences that can occur during shutdown operations. The first sequence begins when an idle-down option is selected; the application program receives an advisory shutdown message, shuts down its connections gracefully, and terminates network access without additional network or host operator action. The second sequence begins when a disable option is selected; the application program receives a mandatory shutdown message and should not attempt to terminate connections gracefully. The third sequence is a hybrid of the first two; if insufficient time elapses between selection of an idle-down option and selection of a disable option, the application program can terminate some of its connections gracefully, but not all of them.

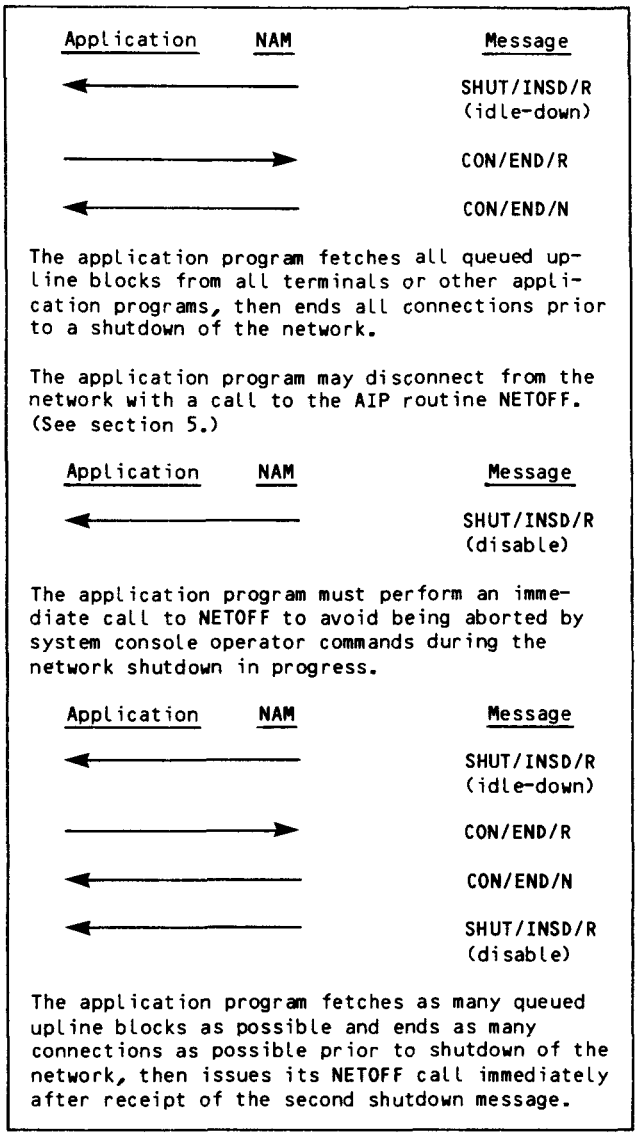
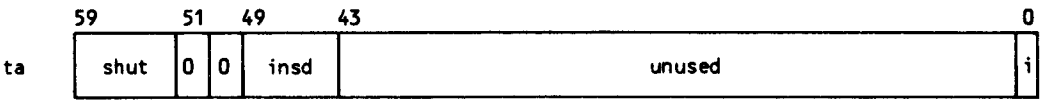


Figure 3-54. Host Shutdown Message Sequences

The Network Access Method does not attempt to force the termination of applications that do not call NETOFF in response to an idle-down or disable request. Normal termination of network operations, however, depends on correct application behavior. Applications that do not eventually call NETOFF after receiving an idle or disable request must be dropped by the host console operator. This then permits normal termination of the network software.

Figure 3-55 shows the two forms of the host-shutdown supervisory message. The application program does not issue a response to this supervisory message.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

shut — Primary function code 42_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol SHUT.

insd — Secondary function code 6. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol INSD.

i — Indicator for type of shutdown message. This field can have the values:

- 0 — This is a normal (warning) message of a pending network shutdown. The network software will not permit any more logical connections to be established, but the application program can inform existing connections of the shutdown, fetch queued input data from all connections, and voluntarily end all connections before issuing a NETOFF call. (See section 5.)
- 1 — Network shutdown is beginning. The application cannot send or receive blocks on any existing connection and no more logical connections can be established. The application program must issue a NETOFF call immediately without ending any existing connections. (See section 5.)

This field can be accessed with the reserved symbol SHUTF as described in section 4.

Figure 3-55. Host-Shutdown (SHUT/INSD/R) Supervisory Message Format

ERROR REPORTING

The primary mechanism used by the network software to indicate logic errors to an application program is an asynchronous supervisory message. In all cases, the message sequence for this mechanism consists of a single message (figure 3-56). The message used in this sequence is the logical-error supervisory message, shown in figure 3-57. The application program does not send a response to this supervisory message.

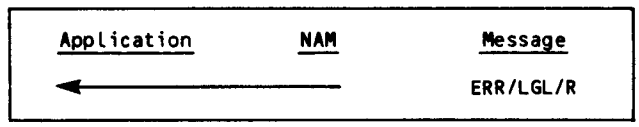
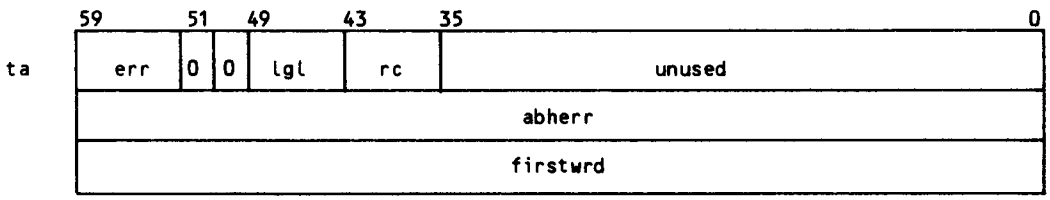


Figure 3-56. Logical Error Message Sequence

As indicated by the reason codes included in the message, many conditions are considered to be logical errors by the network software. The simpler conditions are completely defined within the figure; more details are described here.



ta — Symbolic address of the application program's text area receiving this asynchronous supervisory message.

err — Primary function code 84_{16}. This field can be accessed with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol ERR.

Figure 3-57. Logical-Error (ERR/LGL/R) Supervisory Message Format (Sheet 1 of 2)

lgl — Secondary function code 1. This field can be accessed with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol LGL.

rc — Reason code identifying the cause of the message. This field can contain the values:

1	An invalid act value was specified in the block header of a downline data block or in a DC/CICT/R message.
2	An invalid tlc was encountered; either the value in the block header of a downline block was greater than 2043, or the length of the block exceeded 410 central memory words.
3	An invalid abt value was specified in the block header of a downline block; either the value was 0 or greater than 3.
4	An invalid acn value was encountered in the block header of a downline data block, in a synchronous supervisory message, or in an asynchronous supervisory message.
5	The application block limit of the connection has been exceeded for downline transmissions.
6	More than 100 ERR/LGL/R messages have been issued to the application program, and the program still has upline synchronous supervisory messages queued for it. Until the application program fetches all queued supervisory messages, all downline asynchronous supervisory messages causing ERR/LGL/R messages are ignored.
7	An illegal or illogical supervisory message was encountered; either the combined primary and secondary function codes of the message are not a valid value, or the message is not permitted as part of supervisory message sequences currently in progress with the application program, or a synchronous supervisory message on connection 0.
8	A fragmented input or output error has occurred; a call to NETPUTF, NETGETF, or NETGTFL causes this supervisory message when the block involved in the call contains more than 40 fragments, contains a fragment of more than 63 words, or the total block length in words is inconsistent with the call's tlmax parameter or the block header's tlc value.
9 thru 11	Reserved by CDC for network software use.
12	An application is not allowed to send data blocks on a connection it has established with a passive device.
13 thru 15	Reserved by CDC for network software use.

This field can be accessed with the reserved symbol RC, as described in section 4.

abherr — Application block header word associated with the block or supervisory message that caused the ERR/LGL/R message. This field contains a zero word if the network software cannot provide a copy of the block causing the program. This field can be accessed with the reserved symbol ERRABH, as described in section 4.

firstwrd — The first 60 bits of the block or supervisory message causing the ERR/LGL/R message are placed in this field if the network software can supply the information. This field contains a zero word unless the abherr field is nonzero and the tlc field within that block header is nonzero. This field can be accessed with the reserved symbol ERRMSG, as described in section 4.

Figure 3-57. Logical-Error (ERR/LGL/R) Supervisory Message Format (Sheet 2 of 2)

The rc field value of 1 is received when:

On an application-to-application connection, the application connection specified an application character type of 4 either in the abh or in the change-input-character-type supervisory message.

For a supervisory message the application specified an application character type other than 1, 2, or 3 in the abh.

On an application-to-terminal connection, an application character type other than 2, 3, or 4 was used in a downline block header or a change-input-character-type supervisory message.

The rc field value of 4 is received when:

The application connection number involved is out-of-range for the application program and therefore nonexistent. Connection numbers not yet assigned to the application program, or greater than maxacn, are out of range.

Application connection number 0 is specified in a change-connection-list or turn-list-processing-off supervisory message.

The rc field value of 5 is received when the application program is not using a flow control monitoring mechanism, such as that described earlier in this section. The downline block causing the block limit to be exceeded is discarded. The application program should not transmit any more downline blocks until it has received at least one block-delivered message upline.

The rc field value of 6 is received when the network software attempts to protect itself from application program flaws in supervisory message processing logic. A partial limit imposed on the number of logical errors permitted for an application program prevents the application program from deadlocking the network in such cases. This limit applies only to logical-error messages queued for the application program. The limit keeps the program from committing large numbers of errors in downline transmissions without periodically fetching asynchronous supervisory messages sent upline to identify the errors. The limit is implemented as follows:

Each time the network software sends an asynchronous logical-error message to the application program, a limit counter for the program is incremented by one.

Each time the application program fetches all queued asynchronous supervisory messages it has outstanding, the limit counter for the program is reset to zero.

When the limit counter for the program reaches 100, a logical-error message with the rc field value of 6 is queued for the program. Until the application program fetches all queued asynchronous supervisory messages it has outstanding, any downline transmission by the program that causes a logical-error message condition is discarded by the network software without being processed.

When the limit counter reaches 100, additional asynchronous supervisory messages might already be buffered by AIP. In this case, the maximum number that must be fetched to clear the counter may be as high as 121.

APPLICATION INTERFACE DESCRIPTIONS 4

This section describes the language interface requirements of an application program, the interfacing utilities available to a program, and those aspects of network software internal interfacing that affect program use of certain Network Access Method (NAM) features. However, this manual does not attempt to describe all network software interfaces. Portions of the network software that execute as application programs use supervisory messages that are either not discussed in this manual or else that are modified from the format presented in this manual. This section treats only those areas of interface that are properly used by an installation-written application program.

LANGUAGE INTERFACES

Application program use of the Application Interface Program (AIP) is essentially the same, regardless of the language used to code the application program. Parameter list and calling sequence requirements are the same for COMPASS assembler language and compiler-level languages. The residence of the AIP routines, the form of the calling sequences, and the utilities available to the application program differ for COMPASS and compiler-level languages.

PARAMETER LIST AND CALLING SEQUENCE REQUIREMENTS

The AIP statements and interfacing utilities use standard FORTRAN-style calling sequences and parameter lists; that is, a parameter list contains one 60-bit word per parameter. The address of this parameter list is passed to the appropriate routine in register A1. Linkage with the statement within the application program is performed by executing a return jump instruction (RJ) to the entry point. To provide compact object code, traceback information is not generated, and the parameter list need not be followed by a word of zeros.

Because the statement parameters are passed by address (called by reference), the NAM programmer should be careful about substituting values when defining the parameters. Those parameters identified as return parameters should not be specified as constants or expressions in the call statement. Such specifications can produce unpredictable errors in program code. This restriction is compatible with normal FORTRAN programming practices.

Return parameters are normally defined by variable names, array names, array element names, or similar symbolic addresses. Since the terminology for such entities varies according to the programming language used, this manual uses the term symbolic address for all such possibilities. Unless otherwise stated, numeric absolute or relative addresses are not used in call statements.

Those parameters identified as input parameters can be defined by constants, expressions that can be evaluated to produce constants, or symbolic addresses (as defined above). Input parameters are usually defined by constants or expressions; this manual uses the term value for all such possibilities.

All AIP statement parameters used by a COBOL program must be described in the Data Division as level 01 data entries, or data entries at other levels when the entries are left-justified to word boundaries. COBOL 5 programs that access fields within parameters must also describe the fields in the Data Division as COMP-4 numeric data entries to manipulate values within the fields as 6-bit entities. Direct field access and AIP use is difficult using COBOL; COMPASS macros or FORTRAN subroutines are sometimes necessary to set up parameters before AIP calls or to unpack them after AIP calls.

All direct calls from a COBOL program to AIP must be coded as calls to FORTRAN-X subroutines. Refer to section 5. Indirect use of AIP by a COBOL program is also possible; refer to the Queued Terminal Record Manager description later in this section.

The NAM calling sequence does not permit recursive calls.

PREDEFINED SYMBOLIC NAMES

The fields in NAM supervisory messages have been assigned symbolic names so that they can be identified to the utilities described later in this section. These names are display-coded Hollerith characters and are listed and defined in table 4-1. The capitalized symbol appears as it should be used in calls to NFETCH or NSTORE. The symbols are arranged alphabetically within the table.

TABLE 4-1. RESERVED SYMBOLS

Symbol	Entity Defined by Symbol	Predefined Integer Value
ABHABN	Application block number field in application block header for all upline or downline blocks	None
ABHABT	Application block type field in application block header for all upline or downline blocks	None
ABHACT	Application character type field in application block header for all upline or downline blocks	None
ABHADR	Process number address field in application block header for supervisor program upline or downline blocks (system use only). Application connection number field in application block header for all application program upline or downline blocks.	None
ABHBIT	Parity error flag bit in application block header for upline (input) blocks. Auto-input mode flag bit in application block header for downline (output) blocks.	None
ABHBRK	Break occurred in downline (output) transmission bit in application block header for upline (input) message block.	None
ABHCAN	Cancel previous blocks bit in application block header for upline (input) blocks. Punch banner (lace) card bit in application block header for downline (output) blocks.	None
ABHIBU	Input block undeliverable bit in application block header for upline (input) blocks	None
ABHNFE	No format effectors flag bit in application block header for downline (output) blocks	None
ABHTLC	Text-length-in-character-units field in application block header for all upline or downline blocks	None
ABHTRU	Truncation occurred bit in the application block header for upline (input) data or supervisory message blocks	None
ABHUBF	User break flag	None
ABHWORD	Application block header word for all upline or downline blocks	None
ABHXPT	Transparent mode transmission bit in application block header for all upline or downline blocks	None
ACCON	Character type of CON supervisory messages	1
ACCTRL	Character type of CTRL supervisory messages	2
ACDBG	Character type of DBG supervisory messages	1
ACDC	Character type of DC supervisory messages	1
ACERR	Character type of ERR supervisory messages	1
ACFC	Character type of FC supervisory messages	1
ACHOP	Character type of HOP supervisory messages	1
ACIFC	Character type of IFC supervisory messages	1
ACINTR	Character type of INTR supervisory messages	1
ACK	Secondary function code field for FC/ACK/R	2
ACLST	Character type of LST supervisory messages	1

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
ACRQ	Secondary function code field for CON/ACRQ messages	2
ACSET	Character type of SET supervisory messages	1
ACSHUT	Character type of SHUT supervisory messages	1
ACTCH	Character type of TCH supervisory messages	1
APP	Secondary function code field for INTR/APP messages	2
BRK	Secondary function code field for FC/BRK/R	0
CB	Secondary function code field for CON/CB/R	5
CHAR	Secondary function code field for CTRL/CHAR	8_{16}
CICT	Secondary function code field for DC/CICT/R	0
CON	Primary function code field for connection management messages	63_{16}
CONAABN	Application block number of CON/ACRQ supervisory message	None
CONAAWC	User validation control words in CON/REQ/R	None
CONABL	Application block limit field in CON/REQ/R	None
CONABN	Application block number of CON/ACRQ supervisory message	None
CONABZ	Block size in connection management messages	None
CONACN	Application connection number field in connection management messages	None
CONACR	Primary and secondary function code fields for CON/ACRQ/R, including EB and RB fields as zero	6302_{16}
CONACRA	Primary and secondary code fields in CON/ACR2/A including EB field set to 1	6382_{16}
CONACT	Application input character type field in CON/REQ/N	None
CONAHDS	User validation control words in CON/REQ/R	None
CONAHMT	User validation control words in CON/REQ/R	None
CONAHWS	User validation control words in CON/REQ/R	None
CONALN	Application list number field in CON/REQ/N	None
CONANM	Requesting application program name in CON/REQ/R	None
CONASWI	User validation control words in CON/REQ/R	None
CONATWD	Various default terminal characteristics in the connection management messages	None
CONBDD	Break discard data field in CON/REQ/N	None
CONCB	Primary and secondary function code fields for CON/CB/R, including EB and RB fields as zero	6305_{16}
CONDBZ	Downline block size in the connection management message CON/REQ	None
CONDT	Device type field in CON/REQ/R	None
CONEND	Primary and secondary function code fields in CON/END/R, including EB and RB fields as zero	6306_{16}
CONENDN	Primary and secondary code fields in CON/END/N including RB field set to 1	6346_{16}

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
CONFAM	Family name terminal is logged in under connection management messages	None
CONFO	Family ordinal corresponding to CONFN	None
CONHID	Host node number where requested application resides in connection management messages	None
CONICT	Application input character type	None
CONNXP	No transparent data field in CON/REQ/N	None
CONORD	Device ordinal field in CON/REQ/R	None
CONOWNR	Terminal name field in CON/REQ/R	None
CONPAR	First word of parameters in CON/REQ/R	None
CONPL	Page length field in CON/REQ/R	None
CONPW	Page width field in CON/REQ/R	None
CONR	Restricted interactive capability field in CON/REQ/R	None
CONRAC	Reason code field in CON/REQ/N and CON/REQ/A	None
CONRCB	Reason code field in CON/CB/R	None
CONREQ	Primary and secondary function code fields in CON/REQ/R, including EB and RB fields as zero	6300_{16}
CONREQA	Primary and secondary function code fields in CON/ACRQ/A including EB field set to 1	6380_{16}
CONREQN	Primary and secondary function code fields in CON/REQ/N including RB field set to 1	6340_{16}
CONSCT	Synchronous message type field in CON/REQ/R	None
CONSDT	Sub-device type field in CON/REQ/R	None
CONSL	Security limit field in CON/REQ/R	None
CONT	Terminal class field in CON/REQ/R	None
CONTNM	Terminal name field in CON/REQ/R	None
CONUBZ	Upline block size in the connection management message CON/REQ	None
CONUI	User index field in CON/REQ/R	None
CONUSE	User name field in CON/REQ/R	None
CONXBZ	Transmission block size field in CON/REQ/R	None
CTRCHAR	Primary and secondary code fields in CTRL/CHAR/R, including EB and RB fields as zero	$C108_{16}$
CTRDEF	Primary and secondary function code fields in CTRL/DEF/R, including EB and RB fields as zero	$C104_{16}$
CTRL	Primary function code field in terminal control messages	$C1_{16}$
CTRRTC	Primary and secondary function code fields for CTRL/ATC	$C109_{16}$
CTRTCD	Primary and secondary code fields in CTRL/CHAR/R, including EB and RB fields as zero	$C10A_{16}$

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
DB	Secondary function code field in HOP/DB/R	E_{16}
DC	Primary function code field in DC/CICT/R	$C2_{16}$
DCACN	Application connection number field in DC/CICT/R	None
DCACT	Application character type in DC/CICT	None
DCCICT	Primary and secondary function code fields in DC/CICT/R, including EB and RB fields as zero	$C200_{16}$
DCNXP	No transparent data field in DC/CICT/R	None
DCSCT	Synchronous message type field in DC/CICT/R	None
DCTRU	Primary and secondary function code fields in DC/TRU/R, including EB and RB fields as zero	$C201_{16}$
DE	Secondary function code field in HOP/DE/R	F_{16}
DEFF	Secondary function code field in CTRL/DEF/R	4
DU	Secondary function code field in HOP/DU/R	3
EB	Error bit in all supervisory messages	None
ERR	Primary function code field in ERR/LGL/R	84_{16}
ERRABH	Application block header word in ERR/LGL/R	None
ERRLG	Reason code field in ERR/LGL/R	None
ERRLGL	Primary and secondary function code fields in ERR/LGL/R, including EB and RB fields as zero	8401_{16}
ERRMSG	First message text word in ERR/LGL/R	None
FC	Primary function code field in flow control supervisory messages	83_{16}
FCABN	Application block number field in FC/ACK/R	None
FCACK	Primary and secondary function code fields in FC/ACK/R, including EB and RB fields as zero	8302_{16}
FCACN	Application connection number field in flow control supervisory messages	None
FCBRK	Primary and secondary function code fields in FC/BRK/R, including EB and RB fields as zero	8300_{16}
FCET	Error text in FC/BRK message	None
FCINA	Primary and secondary function code fields in FC/INACT/R, including EB and RB fields as zero	8304_{16}
FCINIT	Primary and secondary function code fields in FC/INIT/R, including EB and RB fields as zero	8307_{16}
FCINITN	Primary and secondary code fields in FC/INIT/N including RB field set to 1	8347_{16}
FCNAK	Primary and secondary function code fields in FC/NAK/R, including EB and RB fields as zero	8303_{16}
FCRBR	Reason code field in FC/BRK/R	None
FCRST	Primary and secondary function code fields in FC/RST/R, including EB and RB fields as zero	8301_{16}

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
FDX	Secondary function code field in LST/FDX/R	3
HDX	Secondary function code field in LST/HDX/R	4
HOP	Primary function code field in HOP supervisory messages	$D0_{16}$
HOPDB	Primary and secondary code fields in HOP/DB/R, including EB and RB fields as zero	$D00E_{16}$
HOPDE	Primary and secondary code fields in HOP/DE/R, including EB and RB fields as zero	$D00F_{16}$
HOPDU	Primary and secondary code fields in HOP/DU/R, including EB and RB fields as zero	$D003_{16}$
HOPNOTR	Primary and secondary code fields in HOP/NOTR/R, including EB and RB fields as zero	$D007_{16}$
HOPREL	Primary and secondary code fields in HOP/REL/R, including EB and RB fields as zero	$D00D_{16}$
HOPRS	Primary and secondary code fields in HOP/RS/R, including EB and RB fields as zero	$D008_{16}$
HOPTRCE	Primary and secondary code fields in HOP/TRACE/R, including EB and RB fields as zero	$D002_{16}$
INACT	Secondary function code field in FC/INACT/R	4
INIT	Secondary function code field in FC/INIT/R	7
INSD	Secondary function code field in SHUT/INSD/R	6
INTR	Primary function code field in user-interrupt supervisory messages	80_{16}
INTRACN	Application connection number field in user-interrupt supervisory messages	None
INTRAPP	Primary and secondary function code fields in INTR/APP/R, including EB and RB fields as zerso	8002_{16}
INTRCHR	User interrupt 8-bit ASCII alphabetic character A through Z in typeahead user-interrupt supervisory messages.	None
INTRRSP	Primary and secondary function code fields in INTR/RSP/R, including EB and RB fields as zero	8001_{16}
INTRUSR	Primary and secondary function code fields in INTR/USR/R, including EB and RB fields as zero	8000_{16}
LCONAC	Length in 60-bit words of CON/ACRQ messages	2
LCONACA	Length in 60 bit words of CON/ACRQ/A	2
LCONCB	Length in 60-bit words of CON/CB/R	1
LCONEN	Length in 60-bit words of CON/END/R	2
LCONENN	Length in 60 bit words of CON/END/N	1
LCONREQ	Length in 60-bit words of CON/REQ/R message (not including APARAM)	11_{10}
LCORQR	Length in 60-bit words of CON/REQ/N and CON/REQ/A	1
LCTRL	Length in 60-bit words of terminal control messages	2

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
LDC	Length in 60-bit words of DC/CICT/R	1
LERR	Length in 60-bit words of ERR/LGL/R	3
LFC	Length in 60-bit words of flow control supervisory messages (except FC/BRK)	1
LFCACK	Length in 60 bit words of FC/ACR/R	1
LFCBRK	Length of FC/BRK message	2
LFCINCT	Length in 60 bit words of FC/INACT/R	1
LFCINIT	Length in 60 bit words of FC/INIT/R	1
LFCINITN	Length in 60 bit words of FC/INIT/N	1
LFCNAK	Length in 60 bit words of FC/NAK/R	1
LFCRST	Length in 60 bit words of FC/RST/R	1
LG	Secondary function code field in HOP/LG/R	A_{16}
LGL	Secondary function code field in ERR/LGL/R	1
LHOPDB	Length in 60 bit words of HOP/DB/R	1
LHOPDE	Length in 60 bit words of HOP/DE/R	1
LHOPDU	Length in 60 bit words of HOP/DU/R	1
LHOPNTR	Length in 60 bit words of HOP/NOTR/R	1
LHOPREL	Length in 60 bit words of HOP/REL/R	
LHOPRS	Length in 60 bit words of HOP/RS/R	1
LHOPTRA	Length in 60 bit words of HOP/TRACE/R	1
LINTR	Length in 60-bit words of INTR/USR/R and INTR/RSP/R	1
LLST	Length in 60-bit words of list management supervisory messages	1
LSHUT	Length in 60-bit words of SHUT/INSD/R	1
LST	Primary function code field in list management supervisory messages	$C0_{16}$
LSTACN	Application connection number field in list management supervisory messages	None
LSTALN	Application list number field in list management supervisory messages	None
LSTDIS	Initial half duplex field in LST/HDX/R	None
LSTFDX	Primary and secondary function code fields in LST/FDX/R, including EB and RB fields as zero	$C003_{16}$
LSTHDX	Primary and secondary function code fields in LST/HDX/R, including EB and RB fields as zero	$C004_{16}$
LSTOFF	Primary and secondary function code fields in LST/OFF/R, including EB and RB fields as zero	$C000_{16}$
LSTON	Primary and secondary function code fields in LST/ON/R, including EB and RB fields as zero	$C001_{16}$
LSTSWH	Primary and secondary function code fields in LST/SWH/R, including EB and RB fields as zero	$C002_{16}$

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
LTCH	Length in 60-bit words of TCH/TCHAR/R	1
NAK	Secondary function code field in FC/NAK/R	3
NOTR	Secondary function code field in HOP/NOTR/R	7
OFF	Secondary function code field in LST/OFF/R	1
ONN	Secondary function code field in LST/ON/R and PRU/ON messages	0
PFC	Primary function code field in all supervisory messages	None
PFCSFC	Primary and secondary function code fields in all supervisory messages, including EB and RB fields	None
RB	Response bit in all supervisory messages	None
RC	Reason code field in all supervisory messages	None
REL	Secondary function code field in HOP/REL/R	D_{16}
REQ	Secondary function code field in CON/REQ messages	0
RS	Secondary function code field in HOP/RS/R	8_{16}
RSP	Secondary function code field in INTR/RSP/R	1
RST	Secondary function code field in FC/RST/R	1
RTC	Secondary function code in field in CTRL/RTC/R	9_{16}
SFC	Secondary function code field in all supervisory messages	None
SHUINS	Primary and secondary function code fields in SHUT/INSD/R, including EB and RB fields as zero	4206_{16}
SHUT	Primary function code field in SHUT/INSD/R	42_{16}
SHUTF	Shut down type in SHUT/INSD	
SHUTYP	Type of shutdown field in SHUT/INSD/R	None
SPMSG0 thru SPMSG9	The corresponding word zero through nine of any supervisory message	None
SWH	Secondary function code field in LST/SWH/R	2
TCD	Secondary function code field in CTRL/TCD	A_{16}
TCH	Primary function code field in TCH/TCHAR/R	64_{16}
TCHACN	Application connection number field in TCH/TCHAR/R	None
TCHAR	Secondary function code field in TCH/TCHAR/R	0
TCHPL	Page length field in TCH/TCHAR/R	None
TCHPW	Page width field in TCH/TCHAR/R	None
TCHTCH	Primary and secondary function code fields in TCH/TCHAR/R, including EB and RB fields as zero	6400_{16}
TCHTCL	Terminal class field in TCH/TCHAR/R	None
TRACE	Secondary function code field in HOP/TRACE/R	2

Each symbol consists of the characters identifying its field within a message, combined with characters identifying the specific message or group of messages. For example:

All primary function code fields can be accessed through the symbol PFC.

All fields in messages with the primary function code mnemonic CON begin with CON; the application list number field in such messages is therefore CONALN.

All fields in the application block header word can be accessed through symbols beginning with ABH.

Some symbols are restricted to use in certain contexts. For example, the FORTRAN Extended 4 call:

```
IVAL=NFETCH(0,6LCONEND)
```

or the FORTRAN 5 call:

```
IVAL=NFETCH(0,L"CONEND")
```

returns the primary and secondary code value for the corresponding fields in a CON/END/R message; however, the FORTRAN Extended 4 call:

```
CALL NSTORE(SMTA,6LCONEND,IVAL)
```

or the FORTRAN 5 call:

```
CALL NSTORE(SMTA,L"CONEND",IVAL)
```

causes an error message indicating that the symbol CONEND is unrecognized. The symbol is unrecognized because its context is incorrect. The correct FORTRAN Extended 4 call to store the information is:

```
CALL NSTORE(SMTA,6LPFCSFC,IVAL)
```

or the call:

```
CALL NSTORE(SMTA,6LPFCSFC,6LCONEND)
```

These calls correspond to the FORTRAN 5 call:

```
CALL NSTORE(SMTA,L"PFCSFC",IVAL)
```

or the call:

```
CALL NSTORE(SMTA,L"PFCSFC",L"CONEND")
```

There are no predefined names for the AIP statement parameters described in section 5.

PREDEFINED SYMBOLIC VALUES

Some of the supervisory message fields with predefined symbolic names have predefined values that can be obtained through the utilities described later in this section. Values for such names are given in table 4-1, where the names are listed alphabetically.

You can obtain the value assigned to a given symbolic name in the released version of the network software by using a form of the NFETCH utilities. The NFETCH utilities comprise a macro that can be called by a COMPASS program, and a similar subroutine that can be called by a program written in a high-level language.

Be careful in using names with predefined values; in some instances, a name and corresponding value have been assigned to a group of fields. Choosing a wrong name in a utility call can fill more fields than the programmer intends. The NAM programmer should become familiar with all of the predefined symbolic names before using the interfacing utilities.

COMPASS ASSEMBLER LANGUAGE

Application programs coded in COMPASS use AIP statements that make macro calls. These AIP macros reside in the system text library NETTEXT.

Packing and unpacking supervisory message blocks in a COMPASS program is easily accomplished using the interfacing utilities NFETCH and NSTORE. These field access utilities also reside in the system text library NETTEXT. An application program using either utility must first contain calls to SST and NETMAC.

Application Interface Program Macro Call Formats

For those AIP statement calls with parameters, three forms of the COMPASS macro call are possible:

```
[label] macro-name parameters
```

This is the format of the standard call, which produces the full calling sequence.

```
[label] macro-name {LIST=label         }
                   {LIST=register name }
```

When this format is used, macro expansion assumes that the proper calling parameter block is located at the specified address, loads this address into register A1, and performs the call to the AIP procedure.

```
[label] macro-name parameters, LIST
```

When this format is used, macro expansion produces a parameter block in place but does not generate the call to the AIP procedure; the address of the statement using this form is the address used in the second form.

Use the first form when making a straightforward call to the AIP procedures. Use the second form once the parameter list has been created elsewhere with the third form. This form might save space when procedures are used several times.

Example 1:

```
NETPUT IHA,ITA
```

This statement is a direct call to execute the NETPUT macro with the two symbolic address parameters shown.

Example 2:

```
PUT1 NETPUT IHA,ITA,LIST
```

This statement expands the NETPUT macro and creates the indicated parameter list at symbolic address PUT1 but does not execute NETPUT.

Example 3:

```
NETPUT LIST=PUT1
```

This statement actually executes the NETPUT macro with the parameters in the list expanded at location PUT1.

If a macro call is issued with an error, the COMPASS assembler flags the error and provides an explanation during assembly of the macro. A complete listing of the assembly error messages from AIP-related macros is provided in appendix B.

A summary of all the macro call formats available appears in appendix D.

Field Access Utilities

Two additional macros, NFETCH and NSTORE, are provided to make message field definition and access easier. Application programmers are urged to use these macros as described below. Use of these macros and their related predefined symbolic names will simplify application program conversion under future versions of the network software.

NFETCH Macro

A call to the NFETCH macro returns the contents of a specific field within an array of one or more words that comprise all or part of a supervisory message block. The octal integer value returned by the call is right-justified within the X or B register specified in the call.

The format of the NFETCH macro call is given in figure 4-1.

Execution of NFETCH destroys the contents of registers A5, X5, X6, and the X or B register specified to receive the returned value. Execution of NFETCH requires the application program to contain calls to SST and NETMAC. Placing NETTEXT in the COMPASS control statement defines the NFETCH macro and the symbolic names used as the NFETCH field parameters.

As examples of NFETCH use, consider the following operations.

Example 1:

```
NFETCH MYARRAY,PFC,X1
```

This statement places the value of the primary function code field within MYARRAY into register X1. The primary function code field is identified by the symbolic name PFC.

LOCATION	OPERATION	VARIABLE
[label]	NFETCH	array,field,Xj or Bj

label — Optional address label of the macro call.

array — The address of the first word of the array from which the field value should be obtained. This parameter can be:

- An address label
- The name of a register address
- Zero

If zero is declared, any predefined value for the indicated symbolic name is returned.

field — The predefined symbolic name of the field for which a value should be fetched from the array. The possible contents of field are listed alphabetically in table 4-1.

j — The number of the X or B register which should receive the value fetched from the array. The value is right-justified in Xj or Bj on return from the call. When a B register is used, the field to be fetched must be $\leq$ 18 bits long.

Figure 4-1. NFETCH Macro Call Format

Example 2:

```
SX2         BUFFER
NFETCH      X2,SFC,X3
```

These statements place the value of the secondary function code field within BUFFER into register X3. The secondary function code field is identified by the symbolic name SFC, and the address label BUFFER is supplied through register X2.

Example 3:

```
NFETCH      ARRAY,EB,X3
NZ          X3,ERROR
```

These statements place the value of the error bit (EB) within ARRAY into register X3. If the value in X3 is nonzero (if EB has a value of 1), a jump to ERROR occurs.

Example 4:

```
NFETCH      0,CON,X1
```

This statement returns the predefined value 63_{16} in register X1. The value returned is that of the primary function code field of all connection-request supervisory messages, as identified by the predefined symbolic name CON.

If an NFETCH macro call is issued with an error, the COMPASS assembler flags the error and provides an explanation during assembly of the macro. A complete listing of the assembly error messages from NFETCH is included in appendix B.

NSTORE Macro

A call to the NSTORE macro sets the contents of a specific field within an array of one or more words that comprise all or part of a supervisory message block. The format of the NSTORE macro call is given in figure 4-2.

LOCATION	OPERATION	VARIABLE
[label]	NSTORE	array,field=value

label — Optional address label of the macro call.

array — The address of the first word of the array into which the field value should be placed. This parameter can be declared as an address label or the name of an address register.

field — The predefined symbolic name of the field for which a value should be stored in the array. The possible contents of field are listed alphabetically in table 4-1.

value — The value to be stored in the identified field within the array. This parameter can be:

- A right-justified integer
- A right-justified, zero-filled character string
- A symbolic name with a predefined value (see table 4-1)
- Bj or Xj, where j is the number of an X or B register containing one of the first two possibilities for value above.

Figure 4-2. NSTORE Macro Call Format

Execution of NSTORE destroys the contents of registers A5, A6, X5, X6, X7, and any X or B register specified in the call. Execution of NSTORE requires the application program to contain calls to SST and NETMAC. Placing NETTEXT in the COMPASS control statement defines the NSTORE macro and the symbolic names used as the NSTORE field parameters.

As examples of NSTORE use, consider the following operations.

Example 1:

```
SX2         MYARRAY
NSTORE      X2, PFC=CTRL
```

These statements store the value predefined for CTRL in the primary function code field of MYARRAY. The primary function code field is identified by the symbolic name PFC, and the address label MYARRAY is obtained through register X2.

Example 2:

```
NSTORE      MYARRAY, PFC=CTRL
```

This statement performs the same operation shown in example 1.

Example 3:

```
NSTORE      MYARRAY, CONOWT=7RTERMABC
```

This statement stores the terminal name TERMABC in the owning console terminal name field of MYARRAY. The owning console terminal name field is identified by the predefined symbolic name CONOWT.

If an NSTORE macro call is issued with an error, the COMPASS assembler flags the error and provides an explanation during assembly of the macro. A complete listing of the assembly error messages from NSTORE is included in appendix B.

COMPILER-LEVEL LANGUAGES

Application programs coded in compiler-level languages such as FORTRAN use AIP statements that make relocatable subroutine calls. Such statements need not be declared as external routines. Entry point references are satisfied by the CYBER loader; the AIP routines are loaded from the local library NETIO or NETIOD, which must be declared in an LDSET or LIBRARY control statement.

READ, WRITE, and CONNEC are not employed when NAM is used by a FORTRAN program for input and output between the program and terminals. Terminals serviced by an application program do not have logical unit numbers.

ACCEPT and DISPLAY are not employed when NAM is used by a COBOL program for input and output between the program and terminals you can use. You can use these verbs in COBOL programs that use other network application programs, such as the CDC-written Transaction Facility (TAF), for network access.

Packing and unpacking supervisory message blocks in a compiler-level program is easily accomplished using the interfacing utilities NFETCH and NSTORE. These field access utilities reside in local library NETIO or NETIOD.

Programs written using compiler-level languages can also use the AIP routines indirectly through the utility package called the Queued Terminal Record Manager (QTRM). QTRM is described at the end of this subsection and the use of QTRM is completely defined in appendix E. The subroutines comprising QTRM reside in local library NETIO or NETIOD.

Application Interface Program Subroutine Call Formats

Only one form of the AIP subroutine call is possible in compiler-level language programs. This form is:

```
subroutine-name (parameters)
```

The syntax of this form is discussed in section 5. A summary of all the calls available appears in appendix D. The FORTRAN form of the subroutine call format is the format used throughout this manual when discussing the AIP routines.

Field Access Utilities

Two additional relocatable subroutines, NFETCH and NSTORE, are provided to make message field definition and access easier. Use of these routines and their related predefined symbolic names will simplify application program conversion under future versions of the network software. Because each call to one of these routines causes a table scan, use of the routines increases program execution time. This increase can be minimized by setting up all constants processed by calls to the routines with a single set of calls at the beginning of the program.

NFETCH Function

A call to the NFETCH function subprogram returns an integer value for the contents of a specific field within an array of one or more words that comprise all or part of a supervisory message block. NFETCH can be used anywhere in a program expression that an operand can be used; figure 4-3 defines the format for NFETCH as it is used in an assignment statement.

[ivalue=] NFETCH(array,field)

ivalue= A return parameter; as input to the call, an optional integer variable to receive the value returned for the function.

array An input parameter, specifying the symbolic address of the first word of the array from which the field value can be obtained. This parameter can be:

The array name

Zero

If zero is declared, any predefined value for the indicated symbolic name is returned.

field An input parameter, specifying the predefined symbolic name of the field for which a value should be fetched from the array. The possible contents of field are listed in table 4-1. This parameter must be left-justified with zero fill.

Figure 4-3. NFETCH Integer Function FORTRAN Call Format

The size of the field involved in the NFETCH call determines the format of the content value returned. The field is read as an octal value and the value returned is right-justified as either an integer or a display code character string.

If either the field or array parameter is omitted from the function statement, the application program is aborted and a dayfile message is issued. (See appendix B.)

As examples of NFETCH uses, consider the following operations.

Example 1:

The FORTRAN Extended 4 statement:

```
M=NFETCH(ARRAY,2LEB)
```

or the FORTRAN 5 statement:

```
M=NFETCH(ARRAY,L"EB")
```

makes M equivalent to the value of the error bit. The error bit is identified by the predefined symbolic name EB, left-justified with zero fill in the call.

Example 2:

The FORTRAN Extended 4 statement:

```
M=NFETCH(0,3LCON)
```

or the FORTRAN 5 statement:

```
M=NFETCH(0,L"CON")
```

makes M the integer value 14_{38}, equivalent to the predefined value for the primary function code field in all connection-request supervisory messages. The primary function code field is identified by the predefined symbolic name CON, left-justified with zero fill in the call.

Example 3:

The FORTRAN Extended 4 statement:

```
IF (NFETCH(ARRAY,2LEB).EQ.1) CALL ERROR
```

or the FORTRAN 5 statement:

```
IF(NFETCH(ARRAY,L"EB").EQ.1) CALL ERROR
```

causes a jump to ERROR if the value of the error bit (EB) within ARRAY is 1.

NSTORE Subroutine

A call to the NSTORE subroutine sets the contents of a specific field within an array of one or more words that comprise all or part of a supervisory message block. Figure 4-4 gives the FORTRAN format of the NSTORE call statement.

Integer values stored by the NSTORE call are stored as integers. Character strings are stored in display code form and symbolic names are converted to octal equivalents of their predefined values when stored. Only one field can be specified in each call. A value can be stored in a field any time after the array is declared.

If either the array, field, or value parameters are not declared or are nonexistent, the application program is aborted and a dayfile message is issued. (See appendix B.)

CALL NSTORE(array,field,value)

array	A return parameter; as input to the call, the symbolic address of the first word of the array into which the field value should be placed. This parameter is normally the array name.
field	An input parameter, specifying the predefined symbolic name of the field for which a value should be stored in the array. The possible contents of field are listed alphabetically in table 4-1. This parameter must be left-justified with zero fill.
value	An input parameter, specifying the value to be stored in the identified field within the array. This parameter can be: A right-justified integer value A right-justified, zero-filled Hollerith character string A left-justified, zero-filled symbolic name with a predefined value (see table 4-1).

Figure 4-4. NSTORE Subroutine FORTRAN Call Format

As examples of NSTORE use, consider the following operations.

Example 1:

The FORTRAN Extended 4 statement:

```
CALL NSTORE(ARRAY,3LPFC,3LCON)
```

or the FORTRAN 5 statement:

```
CALL NSTORE(ARRAY,L"PFC",L"CON")
```

stores the predefined value for the primary function code of all connection-request supervisory messages in the primary function code field of ARRAY. The primary function code value is identified by the predefined symbolic name CON and the primary function code field by the predefined symbolic name PFC; both names are left-justified with zero fill in the call.

Example 2:

The FORTRAN Extended 4 statement:

```
CALL NSTORE(ARRAY,6LCONOWT,7RTERMABC)
```

or the FORTRAN 5 statement:

```
CALL NSTORE(ARRAY,L"CONOWT",R"TERMABC")
```

stores the display coded terminal name TERMABC in the owning console terminal name field of ARRAY. The owning console terminal name field is identified by the predefined symbolic name CONOWT, left-justified with zero fill in the call.

Example 3:

The FORTRAN Extended 4 statement:

```
CALL NSTORE(ARRAY,2LRB,1)
```

or the FORTRAN 5 statement:

```
CALL NSTORE(ARRAY,L"RB",1)
```

sets the response bit field in ARRAY to 1. The response bit field is identified by the predefined symbolic name RB, left-justified with zero fill in the call.

Queued Terminal Record Manager Utilities

You can set up a teleprocessing service by interfacing an application program directly with AIP through the subroutine calls described in section 5. This interface requires manipulation of many bit-oriented fields, as described in section 2, and multiple operations to perform a single function, as described in section 3. These protocol requirements can be quite complex, dwarfing the portion of a program's code that actually performs a teleprocessing service when the service itself is very simple.

A FORTRAN programmer can use AIP directly with only minor inconvenience when shifting and masking are required. The NFETCH and NSTORE routines permit a COBOL programmer to bypass most of the shifting and masking problems of direct AIP use, but some remain. Shifting and masking is extremely difficult for a COBOL programmer when NFETCH and NSTORE cannot be used because COBOL constrains field access to fields that are multiples of 6 bits. NFETCH, which is coded as a function and not as a subroutine, is not directly callable from a COBOL program because COBOL does not support functions. To use NFETCH, a COBOL programmer must write a subroutine in another applications language.

The Queued Terminal Record Manager (QTRM) utility package allows compiler language users to remain unaware of AIP protocol requirements. QTRM also allows users of COBOL 5.2 (and later versions) to create teleprocessing service programs using an interface that is oriented to fields defined in multiples of 6 bits.

QTRM is an indirect interface to the network; its use is functionally analogous to directly calling CYBER Record Manager. Using QTRM, an application programmer can send messages to and receive messages from a network of terminals as if the programmer were reading and writing records or files in mass storage. This parallelism is shown in figure 4-5.

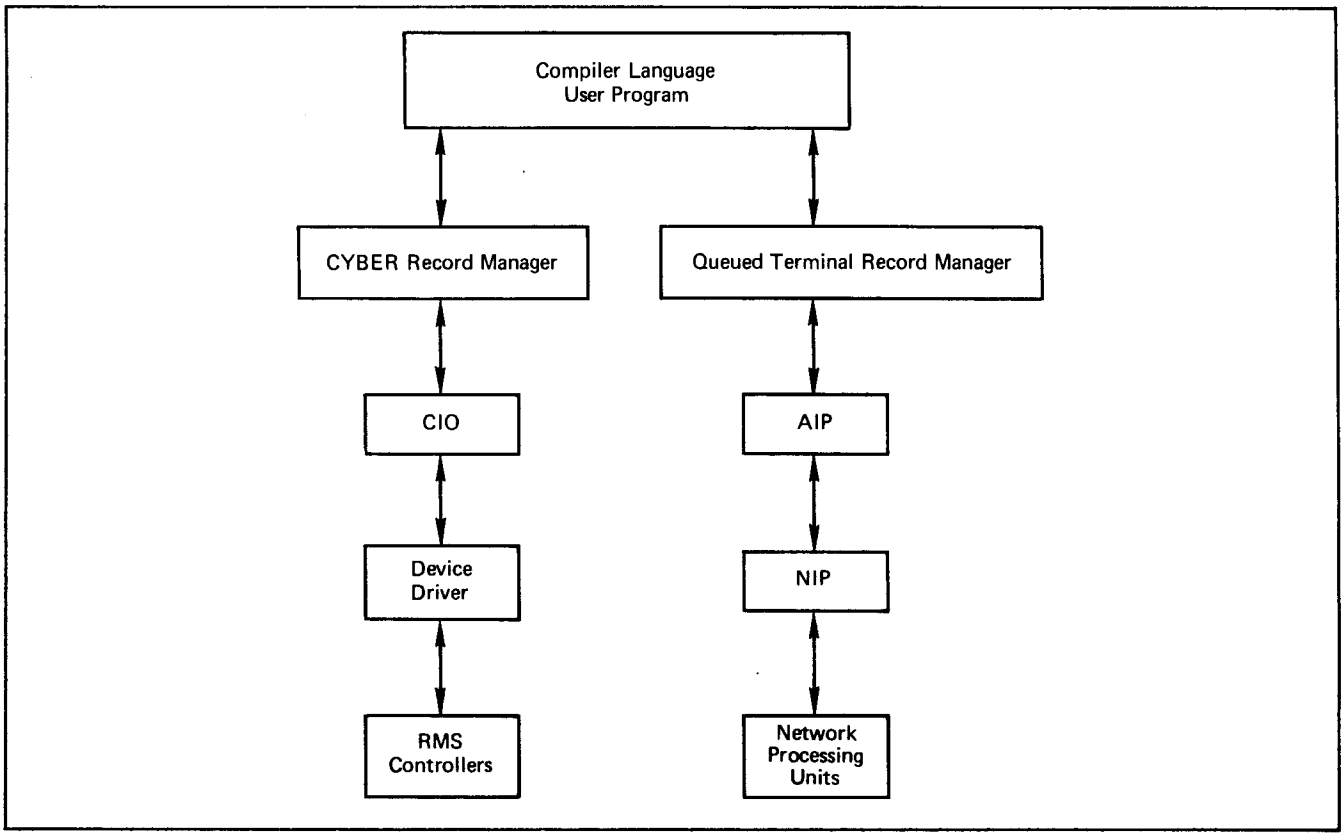


Figure 4-5. QTRM Interface Level Analogy

QTRM is used through calls to the following seven subroutines:

QTOPEN, which is called once to establish communication between the application program and the network. A call to QTOPEN is analogous to opening a mass storage file.

QTLINK, which is called to initiate an application-to-application connection.

QTGET, which is called each time part or all of a message is required from the network. A call to QTGET is analogous to a single read operation on a mass storage file.

QTPUT, which is called each time part or all of a message is intended for the network. A call to QTPUT is analogous to a single write operation on a mass storage file.

QTENDT, which is called to disconnect a single terminal from communicating with the application program.

QTCLOSE, which is called once to disestablish communication between the application program and the network. A call to QTCLOSE is analogous to closing a mass storage file.

QTTIP, which is called to deliver a synchronous supervisory message to a specified connection.

Operation of these procedures is monitored and controlled through a network information table, analogous to a file information table. The network information table contains 10 central memory words of information about each device the application program can potentially service, and 10 words of global information about the state of the application program's communication with the network.

Application programs using QTRM can use only those features of AIP that are provided through the QTRM procedure calls. Such application programs should not also contain calls to AIP routines other than NFETCH and NSTORE. QTRM performs the following functions:

Assigns all active device connections to a single connection list and polls that list for input on behalf of the application program

Performs all asynchronous supervisory message exchanges required during application program execution

Provides the final logical line zero byte terminator in downline blocks containing display code characters

QTRM is a simplified alternative to AIP and therefore does not support all of the AIP features. Features currently not supported by QTRM include the following:

- Parallel mode code execution, as provided through NETSETP and NETCHEK calls
- Fragmented buffer input and output, as provided through NETGETF, NETPUTF, and NETGTFL calls
- Application program connections with passive (batch) devices
- Terminal user interruption of type-ahead processing
- Full-duplex mode
- Runtime selection of debug log file and statistical file entries, as provided through NETDBG and NETSTC calls; both files can be generated or have generation suppressed through selection of the appropriate library during loading of the QTRM routines
- Manipulation of application connection lists, or direct polling of any list as provided through NETGETL and NETGTFL calls
- Use of different application character types for input on the same connection, or on different connections, or change of the application character type used for input during the time the program is connected to the network
- Notification of inactive connections
- Selective polling of input from a specific connection, as provided through NETGET and NETGETF calls
- Transparent mode input
- Disposition of the debug log file during program execution, as provided through the NETREL and NETSETF calls; postprocessing disposition of the file is required
- Transmission of messages to the debug log file, as provided through NETLOG calls
- Exchange package and central memory field length dumps, as provided through NETDMB calls
- Transmission of messages to the statistical log file, as provided through NETLGS calls

Appendix E contains a complete description of the QTRM procedure calls and a sample program illustrating QTRM use by a COBOL programmer. QTRM procedures are not discussed elsewhere because QTRM use precludes direct use of the AIP routines documented by the remainder of this manual.

INTERNAL INTERFACES

The information in the remainder of this section is not needed to create a Network Access Method application program. This information is provided as background for application programmers using the parallel mode processing feature of NAM, and programmers with a need for understanding communication among the components of the network software.

APPLICATION INTERFACE PROGRAM AND NETWORK INTERFACE PROGRAM COMMUNICATION

One copy of the Network Interface Program resides at a control point and communicates with separate copies of the Application Interface Program at each control point containing an application program. Communication between NIP and each copy of AIP occurs through system control point calls initiated by AIP. The mechanism for this communication is a fixed-length buffer of status bits, pointers, and data that is called a worklist.

Worklist Processing

When an application program requests connection with the network, its copy of AIP establishes a long-term connection with NIP. The long-term connection exists until the program requests disconnection from the network, or until NIP is informed of the program's failure or termination by the operating system. While the long-term connection exists, an additional short-term connection occurs whenever AIP initiates a transfer of worklists between itself and NIP. The short-term connection exists until NIP issues a system control point call to end it.

The requests made by an application program to AIP are either satisfied by AIP directly or collected into the worklist contained within the AIP portion of the application program's field length. AIP places entries in this worklist until one of the following occurs, then initiates the short-term connection:

- NETON or NETOFF is called by the application program. (See section 5.)
- The worklist is full.

  Another entry cannot be made without causing the worklist to overflow.
- The application program calls a routine (NETGET, NETGETL, NETGETF, or NETGTFL) that obtains input from the network's data structures, other than AIP queues. (See section 5.)
- NETCHEK is called.
- The application program issues a nonforced NETWAIT call to make itself available for rollout or any input, and no supervisory messages or data are queued for it. (See section 5.)
- The application program issues a forced NETWAIT call.
- The application program calls NETPUTF, unless the total message text involved in the call is small enough to fit in the worklist.

This worklist is used to queue outgoing supervisory and data messages, and to queue incoming (upline) data messages only. A second buffer acts as a queue for incoming supervisory messages. When AIP initiates the short-term connection, it checks to see whether its supervisory message buffer is full; if not, AIP appends a request for supervisory message input to the end of the worklist and passes the worklist to NIP. The period during worklist processing is the only time when NIP can read from or write into the field length of AIP, and then only when AIP initiates the action.

NIP processes the transferred worklist until all of the entries are satisfied, then ends the short-term connection. Worklist processing is suspended when:

The operating system rolls out the application program.

NIP causes the application program to be rolled out in response to the request of the program. (See NETWAIT call, section 5.)

A worklist entry cannot be processed without obtaining additional central memory, which is not available.

Even if there are downline messages queued, no worklist transfer occurs in these instances:

The application program calls a routine (NETGET, NETGETF, NETGETL, or NETGTFL) to obtain asynchronous supervisory messages and AIP transfers any queued messages to the application.

The application program issues a nonforced NETWAIT call and there are supervisory messages or data available for the application.

The application program calls a routine to obtain either asynchronous supervisory messages or data messages and NIP has none queued for the application and AIP has nothing queued for the application.

Generally, an application program does not depend on the status of worklist processing between its corresponding AIP copy and NIP. Most programs can adequately function when concerned only with text area buffers and calls to AIP. However, the Network Access Method does provide a mechanism that allows an application program to monitor worklist processing and execute code dependent on that processing. This mechanism is called parallel mode operation.

Parallel Mode Operation

When an application program issues the call that initiates the long-term connection, it identifies a supervisory status word that is used by AIP as a buffer for several flags. Among the supervisory status word flags are worklist processing bits used during parallel mode operations.

When an application program is not processing in parallel mode (the normal, default condition), its copy of AIP initiates the short-term connection with a system control point call specifying that recall is in effect. In this case, the program's copy of AIP does not regain control of the central processor until all worklist entries are processed by NIP and the short-term connection is ended. Because the application program cannot regain the central processor until its copy of AIP has regained the central processor, the program cannot perform any processing in the interim.

Parallel mode operation is usually beneficial only when used on a dual CPU system, because NIP ordinarily has a higher priority than any application program and gains control of the central processor after a call is made to it. NIP retains control until it completes processing of the worklist request.

Processing in parallel mode is analagous to making operating system calls without recall in effect. An application program enters parallel mode by issuing a call to the AIP routine NETSETP. While in parallel mode, anytime AIP initiates the short-term connection, it does so without specifying recall. The application program's copy of AIP reacquires control of a central processor as soon as the operating system's scheduling algorithm permits, and AIP returns control to the calling point of the application program proper. As long as the short-term connection exists, the application program can continue processing with the sole restriction that it cannot issue calls to any AIP routines other than NETCHEK or NETOFF.

Calls to NETCHEK cause AIP to indicate the current status of worklist processing using a bit in the supervisory status word. After each NETCHEK call, the application program must check the supervisory status word. As soon as the bit indicating completion of worklist processing is set, the program is free to issue any AIP call. Parallel mode processing is ended by a second call to the AIP routine NETSETP.

The worklist processing completion bit serves several purposes in parallel mode operation. Calls to NETCHEK cause this bit to be set when processing of the previous request to AIP has been completed, even when that request did not cause a worklist entry or transfer. When a call to NETCHEK results in the completion bit being set, the application program can:

Safely reuse any header area and text area used in its last AIP call

Assume that any worklist transfer involved in the previous AIP function request resulted in the updating of the other bits in the supervisory status word

When a call to NETCHEK does not result in the completion bit being set, the application program should issue additional NETCHEK calls before executing any code dependent on either condition.

Calls to NETOFF end parallel mode operation by ending both the long-term and short-term connections simultaneously. NIP processes a worklist containing a NETOFF call as if the worklist were transferred while the application program was not processing in parallel mode. Calls to NETCHEK are not necessary to test completion of a NETOFF call.

OTHER SOFTWARE COMMUNICATION

Examination of a complete compiler or assembler listing for an application program will reveal symbols and entry points not discussed in this manual. These symbols and entry points are used internally for interfacing between NIP, AIP, and the operating system. Table 4-2 lists the names of internal procedure calls with an outline of the function of each routine; these calls should not be used directly by the application program. In general, procedure names beginning with the three characters NP$ are reserved for use by AIP and should not be used by application programs. Table 4-3 lists the tables and common blocks involved in the processing of an application program's AIP statements.

The Communications Supervisor, Network Supervisor, and Network Validation Facility interface with NAM via the AIP procedure calls described in section 5. These interfaces use special supervisory messages not described in section 3. These special supervisory messages cannot be used in another NAM application program.

NAM interfaces with the network processing unit software through the Peripheral Interface Program, which uses block protocol. These blocks are compiled or interpreted by NIP.

TABLE 4-2. AIP INTERNAL PROCEDURES

Name	Function
NP$CLK	Used only when AIP is run with either the debugging or statistics option on; gets clock time.
NP$DATE	Used only when AIP is run with either the debugging or statistics option on; gets current date.
NP$DBG	Used only when AIP is run with the debugging option on; makes entries in application program local file ZZZZZDN. These entries show results of calls to other AIP routines by the program. (See section 6.)
NP$DMB	Dumps field length to the ZZZZDMB file.
NP$ERR	Issues error messages to the application program's dayfile.
NP$GET	Creates GET worklist entry to send to NIP.
NP$GSM	Refills AIP's supervisory message buffer. (See worklist processing.)
NP$MSG	Issues dayfile message.
NP$ON	Processes NETON call response from NIP.
NP$OSIF	Issues SSC RA+1 call.
NP$PUT	Creates PUT worklist entry to send to NIP.
NP$PUTF	Creates NETPUTF worklist entry to send to NIP.
NP$RCL	Allows AIP to go into recall.
NP$READ	Used only when AIP is run with the debugging option on; reads job record for NETREL call.
NP$RESP	Processes worklist responses from NIP.
NP$ROUT	Used only when AIP is run with the debugging option on; routes job to input queue for NETREL call.
NP$RTIM	Used only when AIP is run with the debugging option on; gets real time since deadstart.
NP$RWD	Used only when AIP is run with the debugging option on; rewinds a file.
NP$SEND	Called when a worklist must be transferred to NIP.
NP$SLOF	Used only when AIP is run with the debugging option on; executes SETLOF macro for NETSETF call. (See section 6.)
NP$SN	Used only when AIP is run with the statistics option on; accumulates statistical data.

TABLE 4-2. AIP INTERNAL PROCEDURES (Contd)

Name	Function
NP$SPRT	Used only when AIP is run with the statistics option on; makes entries in application program local file ZZZZZSN. (See section 6.)
NP$SYM	Allows COMPASS users access to common symbol definitions.
NP$TIM	Used only when AIP is run with the statistics option on; gets CPU time.
NP$UCV	Used to update AIP control variables.
NP$USI	Used to update the S and I bits in the user communication word.
NP$WRTO	Used only when AIP is run with the debugging option on; writes one word to the AIP debug log file.
NP$WRTR	Used only when AIP is run with either the debugging or statistics option on; writes end-of-record to the debug log file or statistics file.
NT$WRTW	Used only when AIP is run with either the debugging or statistics option on; writes entry to the debug log file or statistics file.
NP$XCDD	Used only when AIP is run with the statistics option on; converts numbers to decimal display code.
NP$XFER	Transfers a worklist to NIP.

TABLE 4-3. AIP INTERNAL TABLES AND BLOCKS

Name	Function
NP$DB	Used only when AIP is run with the debugging option on; contains calling parameters for debugging routine NP$DBG.
NP$GETS	Controls variables used to process GET calls.
NP$LOF	Used only when AIP is run with the debugging option on; parameter block for SETLOF macro. (See section 6.)
NP$MODE	Used to keep track of the state the application is in at any one time.
NP$NWL	Worklist for the application program.
NP$NWNC	Used only when AIP is run with the debugging option on; aids in character conversion.
NP$ONAM	NETON entry for the debug log file.
NP$PUTS	Controls variables used to process PUT calls.
NP$SMB	AIP supervisory message buffer for the application program. This block is included in the last 100_8 words of NP$NWL.
NP$STAT	Used only when AIP is run with the debugging option on; contains statistics gathered by NIP. (See section 6.)
NP$TAA	Used to reference the text area array (TAA) in fragmented GETs and PUTs
NP$ZHDR	Header entry for the debug log file.

This section describes the Application Interface Program (AIP) statements used by a network application program to access the network, control network processing, and transmit and receive the messages described in sections 2 and 3.

SYNTAX

Application Interface Program statements are used in COMPASS programs, or in programs written in high-level languages such as FORTRAN. In most high-level languages, only positional parameters can be used; AIP statements conform to this syntactical requirement and, therefore, do not permit the use of keywords. The interpretation attached to a given parameter is determined solely by its location within the string of parameters of each AIP statement. All input parameters must be supplied; there are no defaults.

The FORTRAN positional form is used throughout this section to present AIP statements. Coding the statements when they are used in other languages requires few modifications. For example, in the form of a COMPASS macro call, a sample NETGETL statement has the form:

```
[label] NETGETL aln, ha, ta, tlmax,[LIST]
```

This converts to the FORTRAN subroutine syntax, which is:

```
CALL NETGETL (aln, ha, ta, tlmax)
```

Use of LIST and label are discussed in section 4 where COMPASS interface requirements are given.

The FORTRAN subroutine syntax, in turn, converts to the following COBOL syntax for the same statement:

```
ENTER FORTRAN-X NETGETL
USING aln,ha,ta,tlmax
```

The mnemonic variables identifying each parameter are defined in the statement descriptions, along with any coding constraints imposed on them. Commas delimit parameters in all languages; the significance of blanks depends on the language used. Unless otherwise specified, all values supplied for parameters should be decimal integers.

General definitions of terms appearing in parameter descriptions are given in the glossary. More detailed definitions and parameter constraints that depend on the programming language used are given in section 4 under the heading of Language Interfaces. Program structural considerations that depend on command use are described in section 6 under the headings of Commands and Dependencies.

NETWORK ACCESS STATEMENTS

An application program uses two AIP statements to begin and end access to the network's resources. The NETON statement must be used before the program can use any other AIP statement except NETREL, NSTORE, NFETCH, NETSETF, NETCHEK, NETSETP, or NETOFF. The NETOFF statement must be used after all AIP functions are completed to cause the AIP portion of the application program to perform vital housekeeping tasks; these tasks are associated with debug log file, statistical file, and login processing by the network software.

CONNECTING TO NETWORK (NETON)

The NETON statement (figure 5-1) performs the following functions:

- Identifies the application program to the network so that the Network Validation Facility (NVF) can validate the right of the program to access the network's resources
- Causes AIP to establish communication with NIP
- Identifies a word to be used for communication from AIP to the program, outside of the supervisory message mechanism (figure 5-2)
- Informs the network software of limitations on the number of logical connections the program can handle
- Causes AIP to begin debug log file and statistical file compilation, if AIP contains code permitting this (See section 6.)

An application program must successfully complete a NETON call before it can use any AIP statement other than NETOFF, NETCHEK, NETREL, NETSETF, or NETSETP. If another AIP statement is used before a NETON call is successfully completed, AIP aborts the job and issues a message to the job's dayfile. The incorrectly placed call has no other effect.

An application program's NETON statement is successfully validated by the Network Validation Facility when the program name contained in the NETON call appears in the system common deck COMTNAP. If the program is defined as a privileged application in the local configuration file, it must meet the residency requirements for such to be successfully validated. (See section 6.) The origin type of the program job does not affect this portion of program validation. If validation is not successful, the application program is aborted. If validation is successful, the program has access to the network as long as a NETOFF statement is not issued and communication with NIP continues.

```
CALL NETON (aname,nsup,status,minacn,maxacn)
```

aname An input parameter, specifying in display code the name of the application program, as it is identified for log in and in the local configuration file. This can be one to seven 6-bit alphabetic and numeric characters, but the first must be alphabetic. This parameter must be left-justified, with blank fill. It is advisable to avoid names beginning with the letters NET to make loader map interpretation easier. The following application program names are reserved for internal networks use:

ALL	IAF	NAM	NVF	PTFU	TAF
BYE	LOGIN	NIP	PFU	QTF	TCF
CS	LOGOUT	NS	PTF	QTFS	TVF
HELLO	MCS	NUL	PTFS	RBF	

Use of some of these names causes the program job to be aborted; use of the remainder can cause unpredictable errors.

nsup A return parameter; as input to the call, nsup is the symbolic address of the supervisory status word for communication from AIP to the application program. This word has the format shown in figure 5-2. The upper bit of this word is relevant during parallel mode processing only; this bit reports the status of worklist processing and is updated after each AIP call except NETSETP. Bits 56 and 55 are set when indicated in the figure to report the status of the data message and supervisory message queuing performed by AIP. These bits are valid after any AIP call except NETDBG, NETLOG, NETREL, NETSETF, NETSETP, or NETSTC. This word need not contain zeros at the time of the NETON call and should not be changed at any time by the application program.

status A return parameter; as input to the call, status is the symbolic address of the NETON call status word. On return from the call (or when worklist processing is complete if the call was made in parallel mode), the content of this word indicates the network software's disposition of the application program's NETON attempt. The values of status can be:

0 NETON was successful.

1 NETON was unsuccessful because NIP was not at a control point or did not have enough resources to service this application program (too many application programs running at the same time).

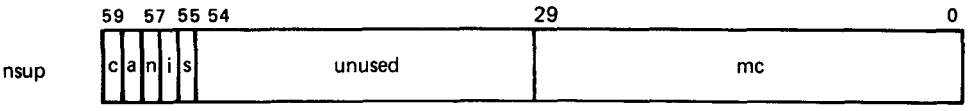
2 NETON was rejected because an application program is presently accessing the network with the same name as specified for the aname parameter.

3 NETON was rejected because the application program has a status of disabled in the Communications Supervisor's tables. The program must be rerun after its entry in the local configuration file has been changed or after the network operator has enabled it.

minacn An input parameter, specifying the smallest application connection number the application program can process; $0 < minacn \leq maxacn \leq 4095$. The network software assigns acn values to connections, beginning with the number specified for minacn. (See section 2.)

maxacn An input parameter, specifying the largest applicaton connection number the application program can process; $0 < minacn \leq maxacn \leq 4095$. The network software does not attempt to complete any more connections to the program after all connections from minacn through maxacn (inclusive) are in use.

Figure 5-1. NETON Statement FORTRAN Call Format



c AIP request and worklist processing completion bit. This bit is relevant only in parallel mode. When any AIP routine other than NETSETP is entered and the AIP function is not completed, the bit is set to zero. If the AIP function is completed, the bit is set to one, if a worklist transfer was required. If the bit is zero, the program cannot call any AIP routines except NETCHEK or NETOFF nor can it use the header area and text area of the last AIP call until the bit is set to one. The bit is set to one by NETCHECK when the last AIP function is completed.

a Reserved for CDC use.

n NAM available bit. This bit is set to one upon return from a NETON call if NAM is available, and zero if NAM is not available. The bit is also set to zero by AIP when AIP is informed by the operating systems that NAM is no longer available.

i Input-in-queue bit. This bit is set to one if NIP has either data messages of synchronous supervisory messages queued for the application. The bit is valid after any AIP call except a call to NETDBG, NETLOG, NETDMB, NETLGS, NETREL, NETSETF, NETSETP, or NETSTC. This bit is set to zero when no data messages or synchronous supervisory messages remain queued for the program.

s Supervisory message in queue bit. This bit is set to one if asynchronous supervisory messages are queued on application connection number 0 for this program. This bit is valid after any AIP call except a call to NETDBG, NETDMB, NETLGS, NETLOG, NETREL, NETSETF, NETSETP, or NETSTC. The s bit is set to zero when no asynchronous supervisory messages remain queued for the program.

mc A count of the number of messages on the debug log file when library NETIOD is used. A NETON call (or a NETREL call with a nonzero lfn parameter value) resets the count to zero (described in section 6).

Figure 5-2. Supervisory Status Word Format

If the program loses communication with NIP, it is aborted by the operating system unless it is a system control point job. System control point jobs are not aborted. The program can reprieve itself from such an abort by using the NOS REPRIEV macro. The program should examine the last error flag that was set for the job (by using the NOS GETJCR macro) to determine the cause of the program's failure. If the program failed because NAM failed, it should issue a NETOFF call and successfully complete another NETON call before issuing any further calls to the AIP routines. The NETOFF call, used in this case, causes AIP to perform internal housekeeping functions and finish information transfer to the debug log and statistical files; the second NETON causes AIP to reinitialize internal tables and reestablish communication with NIP. If a new copy of NIP becomes available prior to the NETOFF call, the second NETON call causes the NETOFF statement to be ignored and program processing can be resumed after new logical connections have been established. Alternating NETON and NETOFF statement sequences in parallel mode have unpredictable results.

The network software tracks an application program and issues dayfile messages concerning the program on the basis of the aname parameter used in the program's NETON call. The operating system, however, is unaware of this name and issues dayfile messages on the basis of the job name assigned to the program according to the contents of the job's command portion. So that all dayfile messages concerning the same program can be identified, you should take the steps described in section 6.

Figure 5-3 contains a portion of a FORTRAN program that correctly performs a NETON call. The program, called PROG, is identified by that name in COMTNAP and in the local configuration file as a nonprivileged application. PROG can process up to three logical connections but requires connections to be numbered beginning with 2. PROG uses the integer word NSUP as a supervisory status word for communication from AIP and tests for successful completion of the NETON call through the integer word NSTATUS.

DISCONNECTING FROM NETWORK (NETOFF)

The NETOFF statement (figure 5-4) performs the following functions:

Breaks AIP communication with NIP

Causes AIP to finish formatting and transferring information for the debug log file and statistical file, if these files are being compiled

Clears AIP internal tables so that the program can issue another NETON call, if necessary

```
      COMMON NSUP,HA(2),TA(200,2)
        •
        •
      NAME=4HPROG
      NSTATUS=0
      MINACN=2
      MAXACN=4
      CALL NETON(NAME,NSUP,NSTATUS,MINACN,MAXACN)
      IF (NSTATUS.NE.0) GO TO 999
        •
        •
  999 PRINT 998, NSTATUS
  998 FORMAT(* NSTATUS IS *, I12)
      STOP
        •
        •
```

Figure 5-3. NETON Statement Example

```
CALL NETOFF
```

Figure 5-4. NETOFF Statement FORTRAN Call Format

The NETOFF statement is used only after all processing of logical connection activities is finished and the program is prepared to end connection with the network. After the NETOFF call is completed, no AIP statement other than NETON, NETREL, NSTORE, NFETCH, NETDMB, and NETSETF can be used. The NETOFF call breaks any logical connection still existing between the application program and a terminal or another application and prevents the network software from attempting to establish any new connection. After the NETOFF statement is processed, the application program continues to execute under control of the operating system.

An application program should always issue a NETOFF call before terminating. Otherwise, the network software informs consoles or other application programs with which connections exist that the program has failed; passive device connections are disposed of by the network software as if the program had failed. Unless a NETOFF call is completed or NETREL is called, the debug log file compiled during job execution cannot be correctly disposed of. Unless a NETOFF call is completed, the statistical file compiled during job execution will not exist.

The NETOFF statement can also be used in a reprieval situation. This use is described under Connecting to Network (NETON).

MESSAGE BLOCK INPUT/OUTPUT STATEMENTS

Input and output on logical connections can be handled through unified or fragmented buffers. Input can be obtained from a connection either by its individual connection number, or according to its membership in a list of connections. AIP statements permit an application program four options for input or output from a specific connection and two options for input from a connection on a list.

SPECIFIC CONNECTIONS

The four options for specific connection input and output are as follows:

- Fetch input to a single, unified buffer (NETGET statement)
- Fetch input to an array of buffers (NETGETF statement)
- Send output from a single, unified buffer (NETPUT statement)
- Send output from an array of buffers (NETPUTF statement)

Inputing to Single Buffer (NETGET)

Each NETGET call transfers one data or supervisory message block from the NIP queue for the connection specified in the call. The NETGET call places the block header in the application program's block header area and the message block in the application program's text area. The NETGET statement has the format shown in figure 5-5.

If no message block is available from the indicated connection, AIP returns a null block; that is, AIP places a header word with an application block type of zero in the header area, and leaves the text area unchanged from what it contained after any previous transfer.

The application program indicates the size of its buffer in each NETGET call. If a message block larger than this size is queued from the specified connection, the message block remains queued. AIP copies the header word of the block into the application program's block header area, sets the ibu bit of the header to one to indicate the condition, and places the actual length of the queued block in the tlc field of the header. The application program's text area is unchanged from what it contained after any previous transfer. To obtain the still-queued message block, the program must issue another NETGET call indicating a buffer size sufficient to accommodate the queued block, or issue a DC/TRU/R asynchronous supervisory message to have the data

```
CALL NETGET(acn,ha,ta,tlmax)
```

acn An input parameter, specifying the application connection number of the logical connection from which a message block is requested. This parameter can have the values:

0	Transfer one asynchronous supervisory message.
minacn ≤ acn ≤ maxacn	Transfer one data block or synchronous supervisory message from the logical connection with the indicated acn.

ha A return parameter; as input to the call, ha is the symbolic address of the application program's header area. The header area always contains an updated application block header after return from the call.

ta A return parameter; as input to the call, the symbolic address of the first word of the buffer array constituting the text area for the application program. On return from the call, the text area contains the requested block if a block was delivered to the application. The text area identified by ta should be at least tlmax words long.

tlmax An input parameter, specifying the maximum length in central memory words of a block the application program can accept. The value declared for tlmax should be less than or equal to the length of the text area identified in the same call; if tlmax is greater than the length of the text area, the block transfer resulting from the NETGET call might overwrite a portion of the program. The maximum value needed for tlmax is a function of the block size used by the connection for input to the program and of the application character type the program has specified for input from the connection. The following ranges are valid:

act=1 $1 \leq tlmax \leq 410$ for 60-bit (one per word) transparent characters

act=2 $1 \leq tlmax \leq 273$ for 8-bit (7.5 per word) ASCII characters

act=3 $1 \leq tlmax \leq 410$ for 8-bit (5 per word) ASCII characters

act=4 $1 \leq tlmax \leq 205$ for 6-bit (10 per word) display code characters

A tlmax value of 0 can be legally declared but results in an input-block-undeliverable condition; that is, an application block header is returned with a set ibu field, even when an empty block of application block type 2 is queued (a block with a tlc value of 0).

Figure 5-5. NETGET Statement FORTRAN Call Format

truncated. (See section 3.) If message truncation is in effect at the time of the NETGET call, then the message will be delivered with the tru bit set in the header.

If the application program's text area is larger than the block transferred by the NETGET call, the portion of the text area after the last word used for the block remains unchanged from what it contained after any previous transfer. If the transferred block does not completely fill the last word used for it, all character positions in the last word used are altered by the transfer. Only the leftmost character positions of the last word included in the block header word tlc field value contain meaningful data.

You can use NETGET to obtain an asynchronous supervisory message from application connection number 0. You can also use NETGET to fetch synchronous supervisory messages and data message blocks from application connection numbers other than 0. Synchronous supervisory messages and data message blocks are never queued on logical connection 0.

Figures 5-6 and 5-7, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, respectively, contain two examples of NETGET use. The first occurrence is in fetching asynchronous supervisory messages, specifically, for connection-request messages. Fetching continues until no asynchronous messages are reported via the supervisory status word (test of NSUP contents). The second appearance of NETGET is in a loop polling for any messages queued on a terminal connection; the polling loop continues until a NETGET call returns a null block. The block header word HA is tested after each call to detect the null block, which has an application block type (ABHABT) of zero.

The value chosen for TLMAX in this example is adequate for both a connection-request supervisory message of ten 60-bit characters and for a logical line of 72 teletypewriter characters with an application character type of 2 used for input. The text area array TA has a dimension of twice TLMAX words, in case the test of ABHIBU fails and a block larger than anticipated must be transferred (third NETGET call).

```
      •
      •
   INTEGER  TA(20),HA,TLMAX,OVTLMAX
   DATA  HA/0/,TA/20*0/,  TLMAX/10/
      •
      •
   NACN=0
 1 CALL  NETGET(NACN,HA,TA,TLMAX)
   IF((NSUP.AND.O"02000000000000000000").EQ.0)
  1GO TO 2
      •
      •
   GO  TO  1
 2 CONTINUE
      •
      •
   NACN=TERM(IACN)
 3 CALL  NETGET(NACN,HA,TA,TLMAX)
   IF(NFETCH(HA,L"ABHABT").EQ.0) GO TO 4
   IF(NFETCH(HA,L"ABHIBU").EQ.1) GO TO 5
 6 CONTINUE
      •
      •
   GO  TO  3
 5 OVTLMAX = NFETCH(HA,L"ABHTLC")/7.5
   ATEMP=NFETCH(HA,L"ABHTLC")/7.5
   IF(ATEMP.NE.OVTLMAX)OVTLMAX=OVTLMAX + 1
   IF(OVTLMAX.GT.20) GO TO 9
   CALL NETGET(NACN,HA,TA,OVTLMAX)
   GO TO 6
 4 CONTINUE
      •
      •
 9 STOP
```

Figure 5-6. NETGET Statement FORTRAN 5 Examples

```
      •
      •
   INTEGER TA(20),HA,TLMAX,OVTLMAX
   DATA HA/0/,TA/20*0/, TLMAX/10/
      •
      •
   NACN=0
 1 CALL NETGET(NACN,HA,TA,TLMAX)
   IF(NSUP.AND.02000000000000000000B).EQ.0)
  1GO TO 2
      •
      •
   GO TO 1
 2 CONTINUE
      •
      •
   NACN=TERM(IACN)
 3 CALL NETGET(NACN,HA,TA,TLMAX)
   IF(NFETCH(HA,6LABHABT).EQ.0) GO TO 4
   IF(NFETCH(HA,6LABHIBU).EQ.1) GO TO 5
 6 CONTINUE
      •
      •
   GO TO 3
 5 OVTLMAX=NFETCH(HA,6LABHTLC)/7.5
   ATEMP=NFETCH(HA,6LABHTLC)/7.5
   IF(ATEMP.NE.OVTLMAX)OVTLMAX=OVTLMAX + 1
   IF(OVTLMAX.GT.20) GO TO 9
   CALL NETGET(NACN,HA,TA,OVTLMAX)
   GO TO 6
 4 CONTINUE
      •
      •
 9 STOP
```

Figure 5-7. NETGET Statement FORTRAN Extended 4 Examples

Inputing to Fragmented Buffer Array (NETGETF)

Each NETGETF call transfers one data or supervisory message block from the NIP queue for the connection specified in the call. The NETGET call places the block header in the application program's block header area. It divides the message block into fragments of whole central memory words and places each fragment in a separately addressed application program text area. The NETGETF statement has the format shown in figure 5-8.

The text areas used are defined for AIP by the text area address array identified in the NETGETF call. This text area address array has the format given in figure 5-9.

The application program indicates the total size of its text area buffers in each NETGETF call through fields in the text area address array. If a message block larger than this total size is queued from the specified connection, the message block remains queued. AIP copies the header word of the block into the application program's header area, sets the ibu bit of the header to one to indicate the condition, and places the actual length of the queued block in the tlc field of the header. The application program's text areas are unchanged from what they contained after any previous transfer. To obtain the still-queued message block, the program must issue another NETGETF call, indicating a total text area size sufficient to accommodate the queued block, or it must issue a DC/TRU/R supervisory message (see section 3).

If the total size of the application program's text areas is larger than the block transferred by the NETGETF call, the portions of the text areas after the last word used for the block remain unchanged from what they contained after any previous transfer. If the transferred block does not completely fill the last word used for it, all character positions in the last word used are altered by the transfer. Only the leftmost character positions of the last word included in the block header word tlc field value contain meaningful data.

If no message block is available from the indicated logical connection, AIP returns a null block; that is, a header word with an application block type of zero is placed in the header area, and the text areas remain unchanged from what they contained after any previous transfer.

You can use NETGETF to obtain an asynchronous supervisory message from application connection number 0. You can also use NETGETF to fetch synchronous supervisory messages and data message blocks from application connection numbers other than 0. Synchronous supervisory messages and data message blocks are never queued on logical connection 0.

```
CALL NETGETF(acn,ha,na,taa)
```

acn — An input parameter, specifying the application connection number of the logical connection from which a message block is requested. This parameter can have the values:

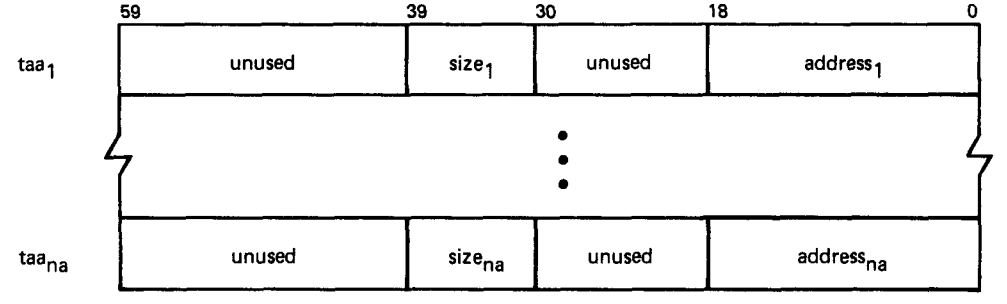
0	Transfer one asynchronous supervisory message.
minacn ≤ acn ≤ maxacn	Transfer one data block or synchronous supervisory message from the logical connection with the indicated acn.

ha — A return parameter; as input to the call, ha is the symbolic address of the application program's header area. The header area always contains an updated application block header after return from the call.

na — An input parameter, specifying the number of fragments the message block should be divided into. The number used should be the same as the number of central memory word entries in the text area address array identified by the taa parameter; if na is greater than the length of the text area address array, the block transfer resulting from the NETGETF call might overwrite a portion of the program. Parameter na can have values $1 \leq na \leq 40$.

taa — An input parameter, specifying the symbolic address of the first word of the one-dimensional array defining the application program's text areas. The array identified by taa has the format shown in figure 5-9.

Figure 5-8. NETGETF Statement FORTRAN Call Format



taa_1 — The symbolic address of the beginning of the array used in the NETGETF call.

$size_i$ — The length in central memory words of block fragment i. This field can contain the values $1 \leq size_i \leq 63$. The sum of all na values for $size_i$ defines the size in central memory words of the largest block the call can transfer; this sum is the equivalent of the tlmax parameter in the NETGET statement. The sum of all na values for size can be 0, but this results in an input-block-undeliverable condition; that is, an application block header is returned with a set ibu field, even when an empty block of application block type 2 is queued (a block with a tic value of 0).

$address_i$ — The relative numeric address of the first word of the application program text area to receive block fragment i. The text area addresses given in this field need not be for contiguous central memory areas.

Figure 5-9. NETGETF Statement Text Area Address Array

Figures 5-10 and 5-11, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, each contain examples of NETGETF use. The program uses the first NETGETF call to fetch a block containing an entire screen of data, which AIP fragments into 12 text areas containing one 60-character physical line each. The application character type chosen for input from the logical connection is 4. The program continues to fetch full screen buffers of data until a null block is encountered by the test of ABHABT. The text areas used are 12 separately addressed 6-word arrays (LINE1 through LINE12), which initially contain blanks (DATA statements). The text area address array (TAA), contains 12 corresponding words; each word contains the relative address of a text area, obtained with the LOCF function. Although the array TAA has a dimension of 24, only the first 12 entries are expected to be used; therefore, a value of 12 is assigned to NA in its DATA statement. Only the first assignment statement constructing TAA is shown; because each text area will contain six words of ten 6-bit characters each, a size of 6 is declared in each TAA entry.

```
      •
      •
   DIMENSION LINE 1(6), . . . ,LINE24(6)
   INTEGER HA,TAA(24),OVRFLNA,TERM(20)
   DATA NA/12/,HA/0/,LINE1/6*L" "/, . . . ,LINE24/6*L" "/
      •
      •
   TAA(1)=SHIFT(6,30).OR.LOCF(LINE1)
      •
      •
   NACN=TERM(IACN)
 1 CALL NETGETF(NACN,HA,NA,TAA)
   IF(NFETCH(HA,L"ABHABT").EQ.0) GO TO 2
   IF(NFETCH(HA,L"ABHIBU").EQ.1) GO TO 5
 6 CONTINUE
      •
      •
   GO TO 1
 5 OVRFLNA=NFETCH(HA,L"ABHTLC")/60.0
   ATEMP=NFETCH(HA,L"ABHTLC")/60.0
   IF(ATEMP.NE.OVRFLNA)OVRFLNA=OVRFLNA + 1
   IF(OVRFLNA.GT.24) GO TO 9
   CALL NETGETF(NACN,HA,OVRFLNA,TAA)
   GO TO 6
 2 CONTINUE
      •
      •
 9 STOP
```

Figure 5-10. NETGETF Statement FORTRAN 5 Examples

```
      •
      •
   DIMENSION LINE 1(6), . . . ,LINE24(6)
   INTEGER HA,TAA(24),OVRFLNA,TERM(20)
   DATA NA/12/,HA/0/,LINE1/6*1L /, . . . ,LINE24/6*1L /
      •
      •
   TAA(1)=SHIFT(6,30).OR.LOCF(LINE1)
      •
      •
   NACN=TERM(IACN)
 1 CALL NETGETF(NACN,HA,NA,TAA)
   IF(NFETCH(HA,6LABHABT).EQ.0) GO TO 2
   IF(NFETCH(HA,6LABHIBU).EQ.1) GO TO 5
 6 CONTINUE
      •
      •
   GO TO 1
 5 OVRFLNA=NFETCH(HA,6LABHTLC)/6.0
   ATEMP=NFETCH(HA,6LABHTLC)/60.0
   IF(ATEMP.NE.OVRFLNA)OVRFLNA=OVRFLNA + 1
   IF(OVRFLNA.GT.24) GO TO 9
   CALL NETGETF(NACN,HA,OVRFLNA,TAA)
   GO TO 6
 2 CONTINUE
      •
      •
 9 STOP
```

Figure 5-11. NETGETF Statement FORTRAN Extended 4 Examples

The second NETGETF call recovers a block not delivered by the original call because the block was larger than expected. This condition is detected by the test of ABHIBU, as returned by the first NETGETF call. The second call is issued with more of the text area address array specified, so that all 24 text areas potentially can be used.

Outputing From Single Buffer (NETPUT)

Each NETPUT call requests AIP to form a message block from the information located in the application program's block header and text areas. The calling application program must construct a complete message block header, as described in section 2. The text portion of the message block can be either a data block, as described in section 2, or a supervisory message block, as described in section 3. The block formed by AIP is sent to the logical connection specified in the block header. The NETPUT statement has the format shown in figure 5-12.

```
CALL NETPUT(ha,ta)

ha      An input parameter, specifying the
        symbolic address of the application
        program's block header area. The block
        header area must contain a valid block
        header word.

ta      An input parameter, specifying the
        symbolic address of the application
        program's text area. The text area must
        contain a valid data message or
        supervisory message block, correctly
        described by the contents of the block
        header area.
```

Figure 5-12. NETPUT Statement FORTRAN Call Format

To reduce data transfer overhead, downline data is sometimes buffered by AIP within the application program's field length. Completion of a NETPUT call therefore does not necessarily mean that the downline data has been transferred to the network.

When an application program is not operating in parallel mode, return from a NETPUT call is equivalent to completion of the call, and the application program can reuse the header area and text area specified in the call immediately. When an application program is operating in parallel mode, return from the call is not equivalent to completion of the call. Completion of the call must be determined through the supervisory status word bits. If completion is not detected when these bits are checked, completion must be forced through calls to NETCHEK. The header area and text area cannot be reused safely until completion occurs. Otherwise, AIP might transfer information on the wrong connection or data other than what the application intended to transfer as part of the block.

Actual transfer of downline data occurs any time the application program makes an AIP call that requires access to the network software's data structures. Any NETGET or NETGETF call causes downline transfers when the call is not made on connection number 0. Any NETWAIT call with a flag value of one causes downline transfers. A NETGETL or NETGTFL call causes downline transfers when the call is not made on list number 0. Other AIP calls do not necessarily cause immediate downline transfers, and downline data buffered by AIP may remain untransferred if the application program is swapped out by the operating system. Downline data buffered by AIP might also remain untransferred if the application program schedules its own central processor usage with the COMPASS macro RECALL, instead of using calls to NETWAIT. To force the transfer of downline data buffered in AIP, call NETCHEK. (See Worklist Processing in section 4.)

You can use NETPUT to send asynchronous supervisory messages to application connection number 0. You can also use NETPUT to send synchronous supervisory messages and data message blocks to application connection numbers other than 0. Synchronous supervisory messages and data message blocks are never sent on logical connection 0.

Figures 5-13 and 5-14, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats respectively, contain an example of NETPUT use. The program has fetched an asynchronous supervisory message and determined that the message is a connection request from a console. The header area contains the connection-request block header. Because asynchronous supervisory messages use an application character type of one, the connection-accepted message being created in the example requires the first NSTORE call to place a 1 in the tlc field. The response message is only one central memory word, viewed as a single character. The next four lines of code modify the first word of the connection-request message, contained in text area TA. First, the NSTORE call sets the response bit (RB). Next, the NSTORE call places a list number in the connection-accepted message, followed by an application character type of 4. Six-bit display code characters are to be used for input from this connection, an option that is legal for consoles because they use the interactive virtual terminal interface. Finally, the NETPUT call sends the completed message on application connection number 0. The incoming block header already contained this number, so the program did not need to supply it while constructing the outgoing block header.

```
      •
      •
CALL NSTORE(HA,L"ABHTLC",1)
CALL NSTORE(TA(1),2LRB,1)
CALL NSTORE(TA(1),L"CONALN",TERM(1,8))
CALL NSTORE(TA(1),L"CONACT",4)
CALL NETPUT(HA,TA)
      •
      •
```

Figure 5-13. NETPUT Statement FORTRAN 5 Example

```
      •
      •
CALL NSTORE(HA,6LABHTLC,1)
CALL NSTORE(TA(1),2LRB,1)
CALL NSTORE(TA(1),6LCONALN,TERM(1,8))
CALL NSTORE(TA(1),6LCONACT,4)
CALL NETPUT(HA,TA)
      •
      •
```

Figure 5-14. NETPUT Statement FORTRAN Extended 4 Example

Outputing From Fragmented Buffer Array (NETPUTF)

Each NETPUTF call requests AIP to form a message block from the information located in the application program's block header and scattered text areas. The calling application program must construct a complete message block header, as described in section 2. The text portion of the message block can be either a data block, as described in section 2, or a supervisory message block, as described in section 3. The block formed by AIP is sent to the logical connection specified in the block header. The NETPUTF statement has the format shown in figure 5-15.

```
CALL NETPUTF(ha,na,taa)

ha      An input parameter, specifying the
        symbolic address of the application
        program's block header area. The block
        header area must contain a valid block
        header word.

na      An input parameter, specifying the number
        of fragments the message block is divided
        into. The number used should be the same
        as the number of central memory word
        entries in the text area address array
        identified by the taa parameter; if na is
        greater than the length of the text area
        address array, the block transferred by
        the NETPUTF call might contain meaningless
        information appended to the last
        meaningful fragment. Parameter na can
        have the values 1 ≤ na ≤ 40.

taa     An input parameter, specifying the
        symbolic address of the first word of the
        one-dimensional array defining the
        application program's text areas. The
        array identified by taa has the format
        shown in figure 5-16.
```

Figure 5-15. NETPUTF Statement FORTRAN Call Format

You can use NETPUTF to send asynchronous supervisory messages to application connection number 0. You can also use NETPUTF to send synchronous supervisory messages and data message blocks to application connection numbers other than 0. Synchronous supervisory messages and data message blocks are never sent on logical connection 0.

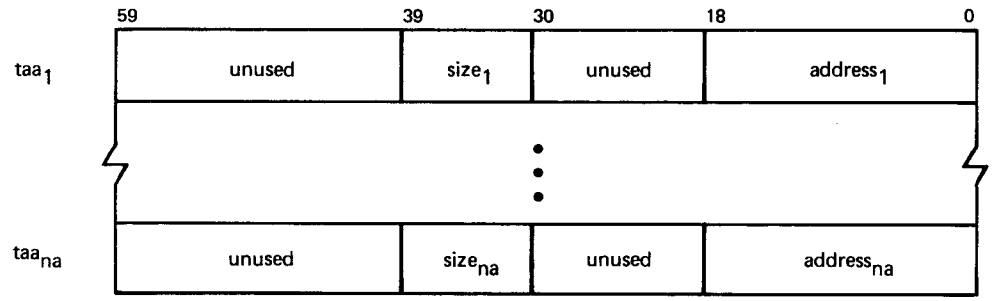
NAM assembles the text portion of the block transferred by the call from separately addressed text areas scattered through the application program's field length. The addresses and sizes of these text areas are supplied to AIP through a text area address array specified in the NETPUTF call. (See figure 5-16.) The total size of all of the text areas identified in the text area array should be greater than or equal to the central memory word equivalent of the number of characters specified in the block header. If the block header declares the block to contain fewer central memory words than all the text areas contain, the portion of the text areas beyond the size declared in the block header will not be included in the transferred block.

To reduce data transfer overhead, downline data is sometimes buffered by AIP within the application program's field length. Completion of a NETPUTF call therefore does not necessarily mean that the downline data has been transferred to the network.

When an application program is not operating in parallel mode, return from a NETPUTF call is equivalent to completion of the call, and the application program can reuse the header area and text areas specified in the call immediately. When an application program is operating in parallel mode, return from the call is not equivalent to completion of the call. Completion of the call must be determined through the supervisory status word bits. If completion is not detected when these bits are checked, completion must be forced through calls to NETCHEK. The header area and text areas cannot be reused safely until completion occurs. Otherwise, AIP might transfer information on the wrong connection or data other than what the application intended to transfer as part of the block.

Actual transfer of downline data occurs any time the application program makes an AIP call that requires access to the network software's data structures. Any NETGET or NETGETF call causes downline transfers when the call is not made on connection number 0. Any NETWAIT call with a flag value of one causes downline transfers. A NETGETL or NETGTFL call causes downline transfers when the call is not made on list number 0. Other AIP calls do not necessarily cause immediate downline transfers, and downline data buffered by AIP might remain untransferred if the application program is swapped out by the operating system. Downline data buffered by AIP might also remain untransferred if the application program schedules its own central processor usage with the COMPASS macro RECALL, instead of using calls to NETWAIT. To force the transfer of downline data buffered in AIP, call NETCHEK. (See Worklist Processing in section 4.)

Figures 5-17 and 5-18, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, respectively, contain an example of NETPUTF use. The program sends a block containing an entire screen of data to an interactive console. AIP assembles the block from text areas containing one logical (and physical) line each. The application character type used for the block is 4. The program uses 12 text areas of separately addressed 6-word arrays (OLINE1 through OLINE12), containing 6-bit display code characters and 12-bit zero byte terminators (DATA statements). The text area address array, OTAA, contains 12 corresponding words; each word contains the relative address of a text area, obtained with the LOCF function. Because the array OTAA has a dimension of 12, a value of 12 is assigned to ONA in its DATA statement. Only the first assignment statement constructing OTAA is shown. Because each text area contains six words of ten 6-bit characters each, a size of 6 is declared in each OTAA entry.



taa_1 The symbolic address of the beginning of the array used in the NETPUTF call.

$size_i$ The length in central memory words of block fragment i. This field can contain the values $1 \leq size_i < 63$. The sum of all na values for size; defines the size in central memory words of the block to transfer; this sum must be less than or equal to 410 central memory words.

$address_i$ The numeric relative address of the first word of the application program text area containing block fragment i. The text area addresses given in this field need not be for contiguous central memory areas.

Figure 5-16. NETPUTF Statement Text Area Address Array

```
      •
      •
DIMENSION OLINE1(6),. . . ,OLINE12(6)
INTEGER HA, OTAA(12), ONA, TERM(20)
DATA ONA/12/,HA/0/,OLINE1/"ABCDEFGHIJ", . . . , L"12345678"/, . . . ,
1DATA OLINE12/"ABCDEFGHIJ", . . . , L"12345678"/
      •
      •
OTAA(1)=SHIFT(6,30).OR.LOCF(OLINE1)
      •
      •
CALL NSTORE(HA, L"ABHABT", 2)
CALL NSTORE(HA, L"ABHADR", TERM(IACN))
CALL NSTORE(HA, L"ABHABN", 1)
CALL NSTORE(HA, L"ABHACT", 4)
CALL NSTORE(HA, L"ABHNFE", 1)
CALL NSTORE(HA, L"ABHTLC", 720)
CALL NETPUTF(HA, ONA, OTAA)
      •
      •
```

Figure 5-17. NETPUTF Statement FORTRAN 5 Example

```
      •
      •
DIMENSION OLINE1(6), . . . ,OLINE12(6)
INTEGER HA, OTAA(12), ONA, TERM(20)
DATA ONA/12/,HA/0/,OLINE1/10HABCDEFGHIJ, . . . ,8L12345678/, . . . ,
1DATA OLINE12/10HABCDEFGHIJ, . . . , 8L12345678/
      •
      •
OTAA(1)=SHIFT(6,30).OR.LOCF(OLINE1)
      •
      •
CALL NSTORE(HA, 6LABHABT,2)
CALL NSTORE(HA, 6LABHADR, TERM(IACN))
CALL NSTORE(HA, 6LABHABN, 1)
CALL NSTORE(HA, 6LABHACT, 4)
CALL NSTORE(HA, 6LABHNFE, 1)
CALL NSTORE(HA, 6LABHTLC, 720)
CALL NETPUTF(HA, ONA, OTAA)
      •
      •
```

Figure 5-18. NETPUTF Statement FORTRAN Extended 4 Example

CONNECTIONS ON LISTS

The two options for input from connections on lists are as follows:

Fetch input to a single, unified buffer (NETGETL statement)

Fetch input to an array of buffers (NETGTFL statement)

Inputing to Single Buffer (NETGETL)

Each NETGETL call causes NAM to select (on a rotating basis) one of the logical connections from a specified list. NAM only chooses a connection that has message blocks queued and that has not been turned off by a LST/OFF/R supervisory message. One message block is transferred from the NIP queue of the selected connection for each call to NETGETL. The NETGETL call places the block header in the application program's header area and the message body in the application's text area. Figure 5-19 shows the format of the NETGETL statement.

Each NETGETL statement causes the connection list to be scanned only once. Scanning begins with the connection immediately following the connection from which a block was previously transferred. The first connection on the list is examined after the last one on the list. Scanning ends when a connection with a queued input block is found. If no connection has a queued input block, scanning ends with the connection preceding the one at which scanning started.

If data or supervisory message blocks are not available from any connection on the list, a null block is returned. A header word with an application block type of zero is placed in the header area, and the text area is unchanged from its content after the last block was obtained. Null blocks are not returned from each connection.

```
CALL NETGETL(aln,ha,ta,tlmax)
```

aln — An input parameter, specifying the number of the connection list to be scanned for a queued block. This parameter can have the values:

0	Obtain all asynchronous supervisory messages queued on application connection number 0 first, then any data or synchronous supervisory message blocks queued on other connections on list zero.
$1 \leq aln \leq 63$	Obtain one data or synchronous supervisory message block from one connection on the indicated list.

ha — A return parameter; as input to the call, the symbolic address of the application program's block header area. The header area always contains an updated application block header word after return from the call.

ta — A return parameter; as input to the call, the symbolic address of the first word of the buffer array constituting the text area for the application program. On return from the call, the text area contains the requested block if a block was available and the text area was large enough. The text area identified by ta should be at least tlmax words long.

tlmax — An input parameter, specifying the maximum length in central memory words of a block the application program can accept. The value declared for tlmax should be less than or equal to the length of the text area identified in the same call; if tlmax is greater than the length of the text area, the block transfer resulting from the NETGETL call might overwrite a portion of the program. The maximum value needed for tlmax is a function of the block size used by the connection for input to the program and of the application character type the program has specified for input from the connection. The following ranges are valid:

act=1	$1 \leq tlmax \leq 410$ for 60-bit (one per word) transparent characters
act=2	$1 \leq tlmax \leq 273$ for 8-bit (7.5 per word) ASCII characters
act=3	$1 \leq tlmax \leq 410$ for 8-bit (5 per word) ASCII characters
act=4	$1 \leq tlmax \leq 205$ for 6-bit (10 per word) display code characters

A tlmax value of 0 can be legally declared but results in an input-block-undeliverable condition; that is, an application block header is returned with an ibu value of 1, even when an empty block of application block type 2 is queued (a block with a tlc value of 0).

Figure 5-19. NETGETL Statement FORTRAN Call Format

The application program indicates the size of its buffer in each NETGETL call. If a message block larger than this size is available for transfer, the message block remains queued, unless message truncation has been requested. AIP copies the header word of the block into the application program's block header area, sets the ibu bit of the header to one to indicate the condition, and places the actual length of the queued block in the tlc field of the header. The application program's text area is unchanged from what it contained after any previous transfer. To obtain the still-queued block, the program must issue a separate NETGET call, indicating a buffer size sufficient to accommodate the queued block, or it may request a truncated message using the DC/TRU/R asynchronous supervisory message (see section 3). The connection pointer within the list is incremented regardless of whether a transfer occurs, so the same connection is not involved in a second NETGETL call.

If the application program's text area is larger than the block transferred by the NETGETL call, the portion of the text area after the last word used for the block remains unchanged from what it contained after any previous transfer. If the transferred block does not completely fill the last word used for it, all character positions in the last word used are altered by the transfer. Only the leftmost character positions of the last word included in the block header word tlc field value contain meaningful data.

You can use NETGETL to obtain an asynchronous supervisory message from application connection number 0. Application connection number 0 is always part of application list number 0. When a NETGETL call specifying input from list 0 is issued, any asynchronous supervisory messages queued for the program are returned before list scanning continues to other connection numbers on list 0. Synchronous supervisory messages and data message blocks on connection numbers other than zero can also be obtained when their connection numbers have been assigned to list 0.

Figures 5-20 and 5-21, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, respectively, contain an example of NETGETL statement use. The program has assigned all interactive consoles to list 0 when accepting connection with them (code not shown). A NETGETL call is used to periodically poll list 0 for asynchronous supervisory messages affecting new or existing connections, and for interactive input affecting passive terminal connections. The TLMAX value of 10 is adequate for

both supervisory messages of application character type 1 and 72-character logical lines in ASCII (application character type 2) from the interactive consoles. Each time list 0 is polled by the NETGETL call, the block header area HA is tested to determine the block type. If a null block (ABHABT of 0) is found, polling ceases. If a block type of 1 or 2 is found, the block is processed (code not shown) and polling continues. If a supervisory message (block type of 3) is found, a subroutine called SMP is entered to process the supervisory message and polling of list 0 continues.

The NETGET call recovers a block not delivered by the original call because the block was larger than expected. This condition is detected by the test of ABHIBU, as returned by the NETGETL call. The NETGET call is issued with more of the text area buffer available; OVTLMAX can be up to twice TLMAX before the text area is completely filled.

Inputing to Fragmented Buffer Array (NETGTFL)

Each NETGTFL call causes NAM to select (on a rotating basis) one of the logical connections from a specified list. NAM only chooses a connection that has message blocks queued and has not been turned off by a supervisory message. One message block is transferred from the NIP queue of the selected connection for each call to NETGTFL; the block header is placed in the application program's header area and the message body is placed in the application's text areas. Figure 5-22 shows the format of the NETGTFL statement.

Each NETGTFL statement causes the connection list to be scanned only once. Scanning begins with the connection immediately following the connection from which a block was previously transferred. The first connection on the list is examined after the last one on the list. Scanning ends when a connection with a queued input block is found. If no connection has a queued input block, scanning ends with the connection preceding the one at which scanning started.

The text areas used are defined for AIP by the text area address array identified in the NETGTFL call. This text area address array has the format shown in figure 5-23.

The application program indicates the total size of its text area buffers in each NETGTFL call through fields in the text area address array. If a message block larger than this total size is queued from the specified connection, the message block remains queued, unless truncation is in effect. (See section 3.) AIP copies the header word of the block into the application program's header area, sets the ibu bit of the header to one to indicate the condition, and places the actual length of the queued block in the tlc field of the header. The application program's text areas are unchanged from what they contained after any previous transfer. To obtain the still-queued message block, the program must issue a separate NETGETF call, indicating a buffer size sufficient to accommodate the queued block. The program also can request data truncation using the DC/TRU/R asynchronous supervisory message. (See section 3.) The connection pointer within the list is incremented regardless of whether a transfer occurs, so the same connection is not involved in a second NETGTFL call.

```
      •
      •
      INTEGER TA(20),HA, TLMAX, OVTLMAX
      DATA HA/0/, TA/20*0/, TLMAX/10/
      •
      •
      NALN=0
    1 CALL NETGETL(NALN,HA, TA, TLMAX)
      IF(NFETCH(HA,L"ABHABT").EQ.0) GO TO 5
      IF(NFETCH(HA,L"ABHABT").NE.3) GO TO 4
      CALL SMP(HA, TA, TLMAX)
      GO TO 1
    4 IF(NFETCH(HA,L"ABHIBU").EQ.1) GO TO 3
    2 CONTINUE
      •
      •
      GO TO 1
    3 OVTLMAX=NFETCH(HA,L"ABHTLC")/7.5
      ATEMP=NFETCH(HA,L"ABHTLC")/7.5
      IF(ATEMP.NE.OVTLMAX)OVTLMAX=OVTLMAX + 1
      IF(OVTLMAX.GT.20) GO TO 9
      NACN=NFETCH(HA,L"ABHADR")
      CALL NETGET(NACN, HA, TA, OVTLMAX)
      •
      •
      GO TO 1
    5 CONTINUE
      •
      •
    9 STOP
```

Figure 5-20. NETGETL Statement FORTRAN 5 Example

```
      •
      •
      INTEGER TA(20),HA, TLMAX, OVTLMAX
      DATA HA/0/, TA/20*0/, TLMAX/10/
      •
      •
      NALN=0
    1 CALL NETGETL(NALN,HA, TA, TLMAX)
      IF(NFETCH(HA, 6LABHABT).EQ.0) GO TO 5
      IF(NFETCH(HA, 6LABHABT).NE.3) GO TO 4
      CALL SMP(HA, TA, TLMAX)
      GO TO 1
    4 IF(NFETCH(HA, 6LABHIBU).EQ.1) GO TO 3
    2 CONTINUE
      •
      •
      GO TO 1
    3 OVTLMAX=NFETCH(HA, 6LABHTLC)/7.5
      ATEMP=NFETCH(HA, 6LABHTLC)/7.5
      IF(ATEMP.NE.OVTLMAX)OVTLMAX=OVTLMAX + 1
      IF(OVTLMAX.GT.20) GO TO 9
      NACN=NFETCH(HA, 6LABHADR)
      CALL NETGET(NACN, HA, TA, OVTLMAX)
      •
      •
      GO TO 1
    5 CONTINUE
      •
      •
    9 STOP
```

Figure 5-21. NETGETL Statement FORTRAN Extended 4 Example

```
CALL NETGTFL(aln,ha,na,taa)
```

aln — An input parameter, specifying the number of the connection list to be scanned for a queued block. This parameter can have the values:

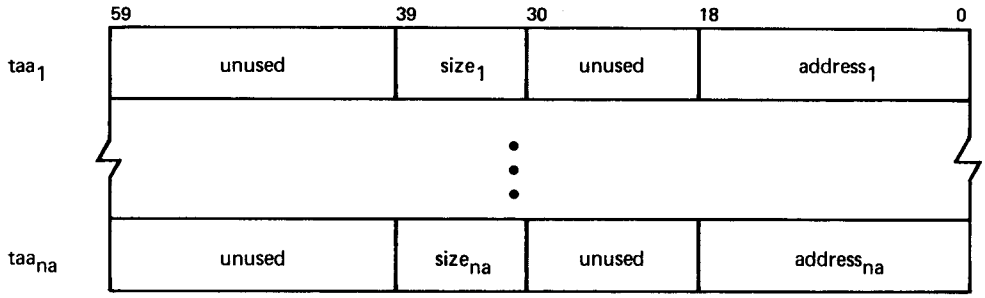
0	Obtain all asynchronous supervisory messages queued on application connection number 0 first, then any data or synchronous supervisory message blocks queued on other connections on list zero.
$1 \leq aln \leq 63$	Obtain one data or synchronous supervisory message block from one connection on the indicated list.

ha — A return parameter; as input to the call, the symbolic address of the application program's block header area. The header area always contains an updated application block header after return from the call.

na — An input parameter, specifying the number of fragments the message block should be divided into. The number used should be the same as the number of central memory word entries in the text area address array identified by the taa parameter; if na is greater than the length of the text area address array, the block transfer resulting from the NETGTFL call might overwrite a portion of the program. Parameter na can have the values $1 \leq na \leq 40$.

taa — An input parameter, specifying the symbolic address of the first word of the one-dimensional array defining the application program's text areas. The array identified by taa has the format shown in figure 5-23.

Figure 5-22. NETGTFL Statement FORTRAN Call Format



taa_1 — The symbolic address of the beginning of the array used in the NETGTFL call.

$size_i$ — The length in central memory words of block fragment i. This field can contain the values $1 \leq size_i < 63$. The sum of all na values for $size_i$ defines the size in central memory words of the largest block the call can transfer; this sum is the equivalent of the tlmax parameter in the NETGETL statement. The sum of all na values for size can be 0, but this results in an input-block-undeliverable condition; that is, an application block header is returned with the ibu field set, even when an empty block of application block type 2 is queued (a block with a tlc value of 0).

$address_i$ — The numeric relative address of the first word of the application program text area to receive block fragment i. The text area addresses given in this field need not be for contiguous central memory areas.

Figure 5-23. NETGTFL Statement Text Area Address Array

If the total size of the application program's text areas is larger than the block transferred by the NETGTFL call, the portions of the text areas after the last word used for the block remain unchanged from what they contained after any previous transfer. If the transferred block does not completely fill the last word used for it, all character positions in the last word are altered by the transfer. Only the leftmost character positions of the last word indicated by the block header word tlc field value contain meaningful data.

If data or supervisory message blocks are not available from any connection on the list, a null block is returned. A header word with an application block type of zero is placed in the header area, and the text areas are unchanged from their contents after the last block was obtained. Null (empty) blocks are not returned from each connection.

You can use NETGTFL to obtain an asynchronous supervisory message from application connection number 0. Application connection number 0 is always part of application list number 0. When a NETGTFL call specifying input from list 0 is issued, any asynchronous supervisory messages queued for the program are returned before list scanning continues to other connection numbers on list 0. Synchronous supervisory messages and data message blocks on connection numbers other than zero can be obtained when their connection numbers have been assigned to list 0.

Figures 5-24 and 5-25, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, respectively, contain an example of NETGTFL use. The program previously assigned all interactive consoles to list 0 when accepting connection with them (code not shown). A NETGTFL call is used to periodically poll list 0 for asynchronous supervisory messages affecting new or existing connections, and for interactive input affecting console connections. If the poll is successful (does not return a null block) and returns an asynchronous supervisory message block, subroutine SMP is called to process the message. If the poll returns a data message block header but no block (test of ABHIBU fails), a NETGETF call is issued with a total text area buffer size larger than in the original call; this NETGETF call should successfully retrieve the queued block.

NAM fragments the block transferred by the NETGTFL or NETGETF call into 12 (NA) or more (OVRFLNA) text areas (LINE1 through LINE24), identified in the 24-entry text area address array (TAA). Each text area is intended to hold one 60-character display coded physical line from a full page of input. NAM places each line into six consecutive central memory words. The calculation of OVRFLNA assumes that an application character type of 4 is used for input, but the size of the LINE1 text area is adequate for both application character type 4 lines and the application character type 1 words used for asynchronous supervisory messages. The FORTRAN function LOCF stores the address of each of the text area arrays in TAA, and the TAA entry has a corresponding length of 6; only the first TAA assignment statement is shown.

```
      •
      •
  DIMENSION LINE1(6), . . . ,LINE24(6)
  INTEGER HA, TAA(24), OVRFLNA
  DATA NA/12/,HA/0/,LINE1/6*L" "/, . . . ,LINE24/6*L" "/
      •
      •
  TAA(1)=SHIFT(6,30).OR.LOCF(LINE1)
      •
      •
  NALN=0
1 CALL NETGTFL(NALN,HA,NA,TAA)
  IF(NFETCH(HA,L"ABHABT").EQ.0) GO TO 5
  IF(NFETCH(HA,L"ABHABT").NE.3) GO TO 4
  CALL SMP(HA,NA,TAA)
  GO TO 1
4 IF(NFETCH(HA,L"ABHIBU").EQ.1) GO TO 3
2 CONTINUE
      •
      •
  GO TO 1
3 OVRFLNA=NFETCH(HA,L"ABHTLC")/60.0
  ATEMP=NFETCH(HA,L"ABHTLC")/60.0
  IF(ATEMP.NE.OVRFLNA)OVRFLNA=OVRFLNA + 1
  IF(OVRFLNA.GT.24) GO TO 9
  NACN=NFETCH(HA,L"ABHADR")
  CALL NETGETF(NACN,HA,OVRFLNA,TAA)
  GO TO 2
5 CONTINUE
      •
      •
9 STOP
```

Figure 5-24. NETGTFL Statement FORTRAN 5 Example

PROCESSING CONTROL STATEMENTS

The three processing control statements NETWAIT, NETSETP, and NETCHEK cause or reduce processing delays to alter the application program's efficiency. These three statements are used in conjunction with the supervisory status word established by the application program in its NETON statement. NETWAIT and NETCHEK can be used by any application program; NETSETP is used only by programs performing parallel mode processing, as described in section 4.

SUSPENDING PROCESSING (NETWAIT)

The NETWAIT statement (figure 5-26) performs the following functions:

Allows an application program to make itself a candidate for rollout by the operating system or otherwise suspend its processing

Allows the application program to declare a maximum time for processing suspension

Allows the application program to delay resumption of processing until input is available for it on any of its logical connections, or on connection zero

Causes the supervisory status word (NETON nsup parameter) for the program to be updated on return from the NETWAIT call

```
      •
      •
   DIMENSION LINE1(6), . . . ,LINE24(6)
   INTEGER HA, TAA(24), OVRFLNA
   DATA NA/12/,HA/0/,LINE1/6*1L /,. . . ,LINE24/6*1L /
      •
      •
   TAA(1)=SHIFT(6,30).OR.LOCF(LINE1)
      •
      •
   NALN=0
 1 CALL NETGTFL(NALN,HA,NA,TAA)
   IF(NFETCH(HA, 6LABHABT).EQ.0) GO TO 5
   IF(NFETCH(HA, 6LABHABT).NE.3) GO TO 4
   CALL SMP(HA,NA,TAA)
   GO TO 1
 4 IF(NFETCH(HA, 6LABHIBU).EQ.1) GO TO 3
 2 CONTINUE
      •
      •
   GO TO 1
 3 OVRFLNA=NFETCH(HA, 6LABHTLC)/60.0
   ATEMP=NFETCH(HA, 6LABHTLC)/60.0
   IF(ATEMP.NE.OVRFLNA)OVRFLNA=OVRFLNA + 1
   IF(OVRFLNA.GT.24) GO TO 9
   NACN=NFETCH(HA, 6LABHADR)
   CALL NETGETF(NACN,HA,OVRFLNA,TAA)
   GO TO 2
 5 CONTINUE
      •
      •
 9 STOP
```

Figure 5-25. NETGTFL Statement FORTRAN Extended 4 Example

Calls to NETWAIT with nonzero flag values always suspend processing when suspension is possible. Calls to NETWAIT with zero flag values suspend processing only when no input is available.

NETWAIT calls with a flag value of 0 should only be made after all outstanding asynchronous supervisory messages have been fetched by the program. A NETWAIT call with a flag value of zero made while any asynchronous supervisory message remains queued always results in immediate return to the program, regardless of whether any other input is available. Such calls represent unnecessary additional processing by AIP and the program and do not cause transfer of worklists that are not completely filled (effectively delaying output resulting from previous calls to NETPUT or NETPUTF).

If NETWAIT is called while the program is operating in parallel mode, parallel mode operation is ignored, and the program is suspended. Parallel mode operation is reinstated when return from the NETWAIT call occurs. You should not issue a call to NETWAIT when it would interrupt parallel mode operation, unless a call to NETCHEK first returns an indication that all worklist processing is completed.

You should include NETWAIT calls in an application program that repeatedly polls the network for input (via NETGET, NETGETL, NETGETF, or NETGTFL calls). If such programs omit frequent NETWAIT calls, severe performance degradation can result; if you perform on-line debugging of such application programs, you should use small time limits for the job while it is in the debugging phase.

```
CALL NETWAIT(time,flag)

time    An input parameter, 1 ≤ time ≤ 4095, specifying the number of seconds for which the application
        program should be suspended. If a value of zero is declared, a default value of one is used;
        if a value greater than 4095 is declared, a default value of 4095 is used.

flag    An input parameter, specifying the conditions under which processing should be resumed. This
        parameter can have the values:

            0   Return from NETWAIT call (resume processing) when input is available from any connec-
                tion, or when the period declared by the time parameter has elapsed. A minimum time
                of 1 second is used if input is not available immediately. When a flag value of zero
                is declared and input is available immediately, the value declared for the time
                parameter is ignored.

            1   Return from NETWAIT call (resume processing) when the period declared by the time
                parameter has elapsed, regardless of whether input is available from any connection.
                Also forces buffer output to be transmitted.
```

Figure 5-26. NETWAIT Statement FORTRAN Call Format

You should use NETWAIT calls as part of the application program's mechanisms to control queuing. For example, the application program must be sure before each NETPUT or NETPUTF call that the call will not cause the logical connection's application block limit to be exceeded. When the limit has been reached, the application program should not output another block until it has received a block-delivered supervisory message for a block already sent. Because repeated polling for supervisory message input to obtain these acknowledgments can degrade program performance, a NETWAIT call should follow any NETPUT or NETPUTF call that might cause the limit to be reached. The time value declared in the NETWAIT call should be large enough to allow a block-delivered supervisory message to be received before another NETPUT or NETPUTF call occurs.

Similarly, an application program should never enter parallel mode after a NETPUT call unless the program first issues a NETWAIT call. Because AIP does not transfer worklists partially filled by NETPUT calls, the NETWAIT call is necessary to force transfer of the worklist. (See Worklist Processing in section 4.) If NETWAIT is not called, the time between the NETSETP call and the first NETCHEK call is not used for network processing.

Figures 5-27 and 5-28, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats, respectively, contain examples of NETWAIT statement use. The program sends a series of data message blocks with NETPUT calls, issues a NETWAIT that transfers the worklist and begins block transmission, and then checks the supervisory status word (NSUP). If no asynchronous supervisory messages are queued on return from the first NETWAIT call, no block-delivered message can have been received and the NSUP test fails. The program issues a second NETWAIT call specifying delay until input on any connection (including the asynchronous supervisory message connection 0) is queued.

```
      MSK1=O"02000000000000000000"
        •
        •
      CALL NETPUT(HA,TA,TLMAX)
      ITIME=1
      IFLAG=1
      CALL NETWAIT(ITIME,IFLAG)
      IF(NSUP.AND.MSK1.EQ.MSK1) GO TO 1
      ITIME=10
      IFLAG=0
      CALL NETWAIT(ITIME,IFLAG)
    1 IACN=0
      CALL NETGET(IACN, HA, TA, TLMAX)
      CALL SMP(HA, TA, TLMAX)
        •
        •
```

Figure 5-27. NETWAIT Statement FORTRAN 5 Examples

```
      MSK1=02000000000000000000B
        •
        •
      CALL NETPUT(HA,TA,TLMAX)
      ITIME=1
      IFLAG=1
      CALL NETWAIT(ITIME,IFLAG)
      IF(NSUP.AND.MSK1.EQ.MSK1) GO TO 1
      ITIME=10
      IFLAG=0
      CALL NETWAIT(ITIME,IFLAG)
    1 IACN=0
      CALL NETGET(IACN, HA, TA, TLMAX)
      CALL SMP(HA, TA, TLMAX)
        •
        •
```

Figure 5-28. NETWAIT Statement FORTRAN Extended 4 Examples

CONTROLLING PARALLEL MODE (NETSETP)

The NETSETP statement (figure 5-29) begins or ends an application program's parallel mode operation. Parallel mode operation involves worklist processing and is discussed in detail under both headings in section 5. While in parallel mode, an application program cannot use any AIP statements other than NETOFF or NETCHEK until AIP processing completion has been indicated in the supervisory status word. The supervisory status word used during parallel mode operation is defined by the nsup parameter in the application program's NETON statement. The bit of the supervisory status word concerned with parallel mode processing is updated only while an application program is operating in parallel mode.

```
CALL NETSETP(option)

option    An input parameter, specifying whether
          parallel mode operation begins or ends
          after the NETSETP call. This parameter
          can have the values:

            =0    Begin parallel mode operation.

            ≠0    End parallel mode operation.
                  (This is the default value for
                  application program operation.)
```

Figure 5-29. NETSETP Statement FORTRAN Call Format

When an application program is operating in parallel mode, it should not alter the contents of the text area used for a NETPUT or NETPUTF call immediately after that call. The program can normally reuse the area as soon as a call to NETWAIT, NETGET, NETGETF, NETGETL, or NETGTFL is completed. The text area used in a NETPUT or NETPUTF call should not be altered until after worklist processing is reported complete; nor should the NETON call status word be tested until then.

A call to NETSETP ending parallel mode operation should not be issued until a call to NETCHEK returns an indication that all worklist processing is completed. AIP ignores calls to NETSETP that attempt to end parallel mode operation if the application program is not operating in parallel mode.

Figures 5-30 and 5-31, which illustrate FORTRAN 5 and FORTRAN Extended 4 formats respectively, contain examples of NETSETP and NETCHEK use. The program attempts to reduce the number of worklist transfers between AIP and NIP to increase its efficiency. It does this while servicing a batch device on application connection number 2 and transmitting to an interactive console on application connection number 3.

```
      •
      •
    ITLMAX=410
    IIACN=3
    IBACN=2
    IOPT=0
    CALL NETSETP(IOPT)
10  DO 99, I = 1, 5, 1
    CALL NSTORE(IIHA(I),L"ABHADR",IIACN)
    CALL NSTORE(IIHA(I),L"ABHABN",I)
    CALL NETPUT(IIHA(I), ITEXT(20*(I-1)))
88  ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.EQ.SHIFT(1, 59)) GO TO 99
    CALL NETCHEK
    GO TO 88
99  CONTINUE
98  ITEMP=NSUP.AND.SHIFT(1, 55)
    IF(ITEMP.EQ.SHIFT(1, 55)) GO TO 3
    ITEMP=NSUP.AND.SHIFT(1, 56)
    IF(ITEMP.EQ.SHIFT(1, 56)) GO TO 4
    ITIME=7
    IFLAG=1
    CALL NETWAIT(ITIME,IFLAG)
    GO TO 98
3   IACN=0
    IOPT=1
    CALL NETSETP(IOPT)
    CALL NETGET(IACN, IHA, ITA, ITLMAX)
      •
      •
4   IOPT=0
    CALL NETSETP(IOPT)
    CALL NETGET(IIACN, IIHA(1), ITEXT(1), ITLMAX)
5   CALL NETCHEK
    ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.NE.SHIFT(1, 59)) GO TO 5
      •
      •
6   CALL NETCHEK
    ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.NE.SHIFT(1, 59)) GO TO 6
      •
      •
    GO TO 10
```

Figure 5-30. NETSETP and NETCHEK Statement FORTRAN 5 Examples

```
      •
      •
    ITLMAX=410
    IIACN=3
    IBACN=2
    IOPT=0
    CALL NETSETP(IOPT)
10  DO 99, I = 1, 5, 1
    CALL NSTORE(IIHA(1), 6LABHADR, IIACN)
    CALL NSTORE(IIHA(1), 6LABHABN, 1)
    CALL NETPUT(IIHA(1), ITEXT(20*(I-1)))
88  ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.EQ.SHIFT(1, 59)) GO TO 99
    CALL NETCHEK
    GO TO 88
99  CONTINUE
98  ITEMP=NSUP.AND.SHIFT(1, 55)
    IF(ITEMP.EQ.SHIFT(1, 55)) GO TO 3
    ITEMP=NSUP.AND.SHIFT(1, 56)
    IF(ITEMP.EQ.SHIFT(1, 56)) GO TO 4
    ITIME=7
    IFLAG=1
    CALL NETWAIT(ITIME,IFLAG)
    GO TO 98
3   IACN=0
    IOPT=1
    CALL NETSETP(IOPT)
    CALL NETGET(IACN, IHA, ITA, ITLMAX)
      •
      •
4   IOPT=0
    CALL NETSETP(IOPT)
    CALL NETGET(IIACN, IIHA(1), ITEXT(1), ITLMAX)
5   CALL NETCHEK
    ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.NE.SHIFT(1, 59)) GO TO 5
      •
      •
6   CALL NETCHEK
    ITEMP=NSUP.AND.SHIFT(1, 59)
    IF(ITEMP.NE.SHIFT(1, 59)) GO TO 6
      •
      •
    GO TO 10
```

Figure 5-31. NETSETP and NETCHEK Statement FORTRAN Extended 4 Examples

The program flow shown minimizes worklist transfers by concentrating the interactive output, instead of interleaving each output line with NETGET calls that might cause worklist transfers by AIP for worklists not completely filled. Parallel mode does not expedite this efficiency, but requirements for its use are illustrated in several parts of the code.

When the program has sent downline all of the blocks it intends to send to the console, it tests for upline data or asynchronous supervisory messages. If neither is found, NETWAIT rolls the program out for 7 seconds and suspends parallel mode processing temporarily.

When asynchronous supervisory messages are found, the program leaves parallel mode processing with a nonzero IOPT parameter in another NETSETP call. The program can then fetch the messages without needing to test NSUP for completion of the NETGET call.

When upline data is found, the program makes sure it is in parallel mode with a zero IOPT parameter in a NETSETP call. This call is ignored if it is reached by a path that had already caused parallel mode processing to begin. While in parallel mode, the program fetches any queued input from the console. NETCHEK is called and tested for completion after the NETGET call. After the attempt to fetch data from the console is completed (the input disposed of by code is not shown), a similar attempt (not shown) is made to fetch data from the batch device. When any batch data has been disposed of, the program returns to its output loop for the console (having presumably prepared the output buffers first).

If a system control point job is operating in parallel mode when it loses communication with NIP, all further network input and ouput AIP calls are ignored, but the program is not aborted. The program should check the n bit in the supervisory status word (see figure 5-2) after completion of all network input and output calls to determine whether or not it is still communicating with NIP.

If a system control point job is not operating in parallel mode when it loses communication with NIP, it is aborted when it makes the next AIP request. The operating system aborts all nonsystem control point jobs when NIP aborts regardless of operating mode.

CHECKING COMPLETION OF WORKLIST PROCESSING (NETCHEK)

The application program uses the NETCHEK statement (figure 5-32) to perform several functions. Each call to NETCHEK:

- Updates bit 59 of the supervisory status word (identified by the nsup parameter used in the NETON statement) on return from the call, when the program is in parallel mode
- Forces AIP to attempt transfer of its current worklist to NIP if the transfer has not yet occurred, if the program is running in either parallel or nonparallel mode

```
CALL NETCHEK
```

Figure 5-32. NETCHEK Statement FORTRAN Call Format

It is not necessary to call NETCHEK to cause worklist transfers. Worklist transfers occur normally after all the requirements described in section 4 under Worklist Processing have been met. A NETCHEK call causes an attempt to transfer a worklist in situations that do not meet these criteria. This operation is equivalent to a NETWAIT except that processing is not suspended.

By checking the supervisory status word after each NETCHEK call, the application program can determine the most recent state of worklist processing and determine whether additional AIP routine calls can be issued. NETCHEK, NETOFF, and NETWAIT are the only AIP statements that can be used while any worklist processing operation is pending. A call to NETSETP ending parallel mode operation should not be issued until a call to NETCHEK returns an indication that all worklist processing has been completed.

If NETON is called during parallel mode operation, NETCHEK should not be called until all worklist processing is reported complete. The NETON call status word does not contain meaningful information until processing for the worklist containing the NETON call is complete. NETCHEK should not be called after a NETOFF call is issued in parallel mode. A NETOFF call ends parallel mode operation by making worklist processing completion status impossible.

Worklist processing is described in section 4. The supervisory status word is described under the heading Connecting to Network at the beginning of this section. Figures 5-30 and 5-31 contain examples of NETCHEK use.

CHARACTERISTICS OF AN APPLICATION PROGRAM 6

This section describes the structure and execution of a Network Access Method (NAM) application program job as a batch or system origin type file.

NOTE

You can create application programs from a time-sharing terminal under NOS, and debug the compiler or assembler logic that way. You can initiate a NAM application from the system console or from the system card reader, via remote batch terminal submittal or via the time-sharing system.

You cannot execute application programs as Transaction Facility tasks.

NOS SYSTEM CONTROL POINT

The NOS system control point facility permits the exchange of data between programs running at different control points. These programs are called:

System control point jobs when they are formally defined as subsystems of the operating system

User control point jobs when they exchange data with a system control point job

System control point jobs (subsystems) can make privileged requests to the operating system and execute with a very high priority. Network system control point jobs such as the Network Interface Program (NIP) usually reside in the operating system library.

Application programs accessing the network execute as system control point jobs or user control point jobs using the system control point facility. Since the code that implements this facility is embedded in the Application Interface Program (AIP), it remains transparent to the application program. Certain aspects of system control point jobs and user control point jobs, however, do affect application program operation.

An application program cannot execute successfully unless the CUCP bit is set in the access word associated with the user name of its job. If the program attempts to access the network and the CUCP bit is not set, the program is aborted with the dayfile messages ILLEGAL USER ACCESS and SYSTEM ABORT, and no error exit processing occurs. Access word bits are set through the MODVAL utility, as described in the NOS System Maintenance reference manual.

While connection to the network exists, a network application program always has a minimum system activity count of one. If the application program uses the control point manager system macro call (GETACT), the minimum system activity count appears in the SCA field of the call. When a network application program ends its connection with the network by a NETOFF call, the system activity count may go down to zero. The GETACT macro is described in volume 4 of the NOS reference set.

APPLICATION JOB STRUCTURE

A batch application program job using the Network Access Method is structured like any other batch job.

A job is a sequence of commands, optionally followed by source programs, object programs, data, or directives. A batch job begins with the job command and ends with an end-of-information indicator. Jobs can consist of either physical card decks or images of card decks.

Application program jobs can enter the system in one of two ways:

Batch jobs on cards are read in through card readers at the central site. Batch jobs of card images are read from a load tape under the direction of the system console operator or the direction of another job.

Remote batch jobs on cards are read in through card readers at remote site terminals. Remote batch job card images are transmitted to form a file at the host computer. All remote batch jobs reach the host computer facilities through the Remote Batch Facility (RBF).

Batch jobs have the same structure no matter what their origin. Remote batch jobs differ from central site batch jobs in that output returns to the terminal and that remote jobs are subject to the limitations of the physical equipment at the remote site. The following information about job decks applies to both card decks and card deck images.

The first card of the batch job deck is the job command; the last card has a 6/7/8/9 multiple punch in column 1. Cards with a 7/8/9 or 6/7/9 multiple punch in column 1 divide the deck into a command portion, program portion, and optional data portion. When a job deck is created as card images from a time-sharing terminal, the cEOR and cEOF entries result in the logical equivalent of 7/8/9 and 6/7/9, respectively. If the job deck is submitted from a HASP or bisynchronous station through the Remote Batch Facility, the /*EORnn and /*EOI cards result in the logical equivalent of 7/8/9 and 6/7/8/9, respectively. HASP or bisynchronous station card readers and card punches support 7/8/9 cards but not 6/7/8/9 cards; 200 User Terminal card readers do not recognize either /*EORnn cards or /*EOI cards.

Jobs in the system waiting to begin execution are collectively known as the input queue. Each job enters the system with the name specified by the first command in the job deck. The operating system changes this name, based on the job command present, to distinguish it from all others in the system.

Once a job enters central memory and begins execution, the image of the job deck is known as a file by the name of INPUT. During job execution, a file with the name of OUTPUT is generated. When the job completes execution, file OUTPUT becomes part of the output queue. The output queue is the collective name for output files remaining in the system when the jobs that generated them have completed execution. As printers, punches, or remote devices become ready, the operating system or remote batch software causes files from the output queue to be physically output. Such files normally return to the user with the system-generated name of the job that created them.

COMMANDS

Commands are instructions to the operating system or its loader. They are grouped together at the beginning of a deck. Collectively, the commands form a job stream. Individually, the commands are job steps.

Commands execute in the order in which they appear in the job stream, unless that order is modified by the operating system control language. Consequently, the order of the commands governs the order of other sections in the deck.

The user is responsible for structuring the job decks so that each command read from file INPUT corresponds correctly with the sections of the job deck. The operating system handles each section of the job deck only once, unless the job specifies contrary handling.

The job command portion of an application program job deck normally contains a USER command as its second card. (See figure 6-1.) The user name specified in this command must have bit 11 (CUCP) of its corresponding access word set, so that the application program can successfully complete calls to system control points. The NOS System Maintenance reference manual describes the mechanism for setting access word bits. Some installations require a CHARGE command following the user statement.

Until the program is successfully compiled, the only other required command is a compiler or assembler execution command in the form described in the appropriate reference manual for the product being used. Figure 6-1 illustrates the use of the compiler execution command for FORTRAN 5. If the job uses a compiler, a LIBRARY or LDSET command is needed to satisfy externals from local libraries NETIO or NETIOD. If the job uses COMPASS, the COMPASS command must declare NETTEXT to satisfy AIP externals and define the symbolic names used for the field access macro utilities NFETCH and NSTORE. (See section 4.)

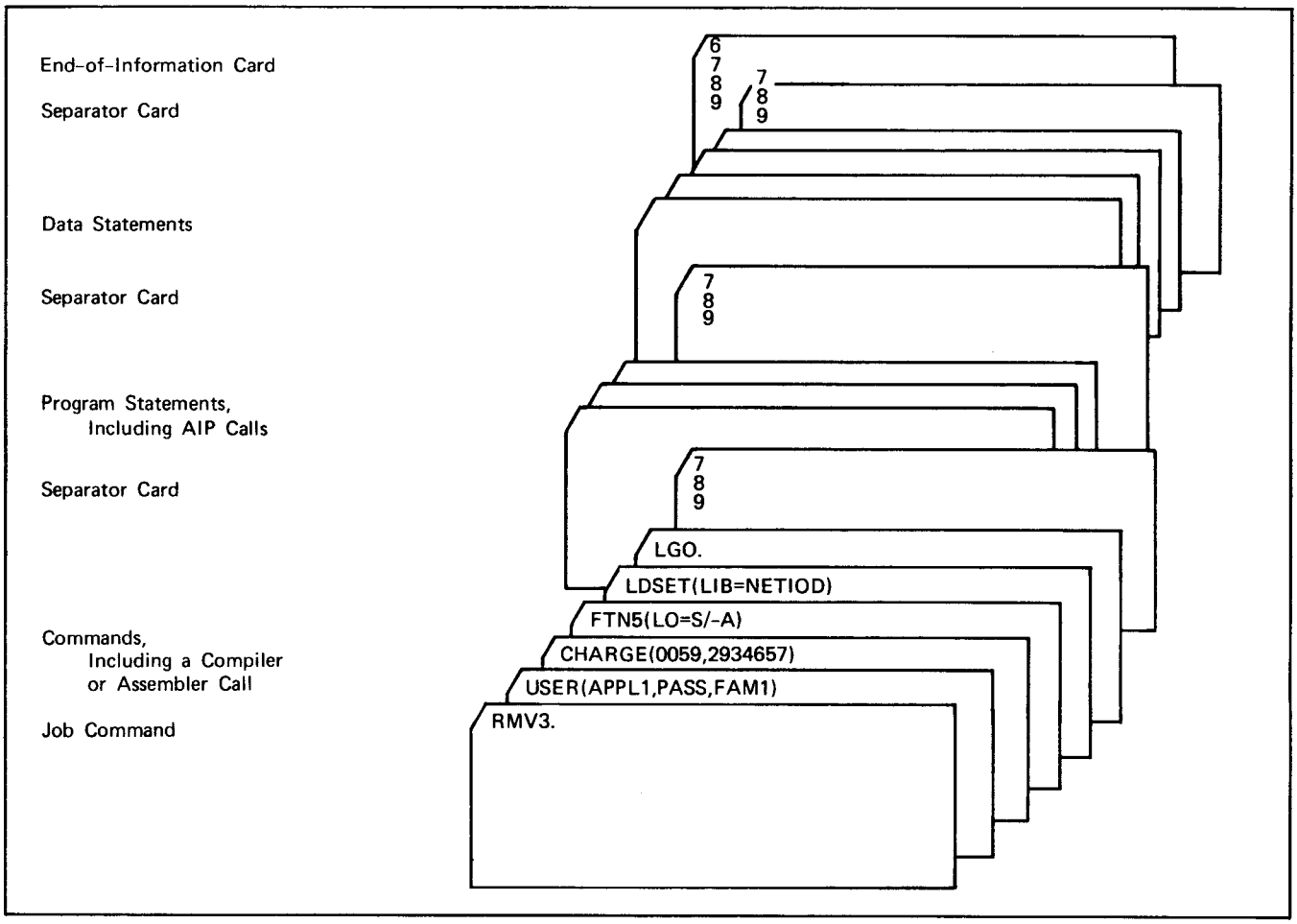


Figure 6-1. Typical Job Structure for System Input

The network software identifies an application program and issues dayfile messages concerning the program on the basis of the aname parameter used in the program's NETON call. The operating system, however, is unaware of this name and issues dayfile messages on the basis of the job name assigned to the program according to the contents of the job's command portion. To ensure that all dayfile messages concerning the application program can be identified, you should take the following steps when the program is run as a batch job:

Determine the method NOS will use to assign a job name to the application program.

Determine the first four characters of that name.

Inform the system console operator of the first characters of the job name corresponding to the application name.

Do not thereafter alter the portion of the job commands that determines the job name.

Alternatively, you can use the NOS control point manager macro GETJN to determine the job name assigned to the application program job during each execution. For the system console operator's information, this name can then be entered in the system dayfile with a message indicating its application program name equivalent. This operation can be performed with the NOS system macro MESSAGE. GETJN and MESSAGE are described in volume 4 of the NOS 2 reference set.

Regardless of the mechanism used to determine the job name, diagnostic messages significant to the system console operator should always be sent to the network operator as well. When the network operator is not usually the system console operator, identification of diagnostic messages by job name association might not be as important as identification of such messages by application program name.

If the job contains commands to reprieve itself from an abort (RERUN or RESTART), the program portion of the job must issue a NETOFF and new NETON call in order to continue accessing the network through NAM.

OVERLAYS

When an application program job is structured to use overlays, the common blocks used by all AIP routines must reside in the main (zero-level) overlay. The operating system loader places the blocks in the main overlay only if the application program makes at least one call to an AIP routine other than NETCHEK in the main overlay. At a minimum, the NETON call must therefore be placed in the main overlay of the program.

ACCESS TO APPLICATION PROGRAMS

Access by terminal users to application programs is controlled in two ways. The first way is by user name and the second way is by terminal device name.

Each user name in the host can be validated to connect to one or any application in the network. This validation is done through MODVAL which is described in the NOS 2 System Maintenance reference manual.

In the local configuration file, each terminal can be designated to have a mandatory or a primary application assigned to it. If the application is mandatory, the terminal cannot be logged into any other application regardless of the user name entered. If the application is primary, the terminal will automatically be connected to the application on the initial login unless an alternate application name is enforced during the login. On subsequent logins the network will prompt for an application and, if a carriage return is entered, the network will connect the terminal to the primary application.

TYPES OF APPLICATION PROGRAMS

All application programs must be configured in the local configuration file (LCF). When an application is defined in the local configuration file it can be declared as having one of the following attributes:

Disabled
Unique Identifier
Privileged

DISABLED

A disabled application is configured in the network, but is not allowed to access the network until the operator enters an enable command to allow it to be connected.

UNIQUE IDENTIFIER

A unique identifier application program requires that interactive terminal user access to it be restricted on the basis of the login parameters used. Only one interactive terminal with a given combination of family name and user index can be connected with a unique identifier application. NVF rejects a terminal user's request to be connected with a unique identifier application if the user logs in with a family and user index combination used by a terminal that is already connected with the application. NVF tells the terminal user to try again later.

As an example, the Remote Batch Facility (RBF) routes its output files on the basis of the family and user names used when the terminal console logs in. So that output will not be misrouted, RBF is normally configured as a unique identifier application program. Thus the family and user index combinations of all interactive terminals accessing the program are guaranteed to be unique.

PRIVILEGED

Privileged application programs must have the CSOJ bit set in the access word associated with the user name for the job executing the program code. This bit provides the program with system origin type permission which is required to access the network successfully.

Jobs with system origin type permission can be executed by system console operator type-in. Such jobs usually reside under the operating system user name in the operating system permanent file catalog or are installed in the operating system library.

Having system origin type permission does not mean that these programs must have a system origin type when executed; rather, a privileged application program is capable of such execution.

Nonprivileged application programs can have any origin type permission. Origin type permission for such programs does not affect access to the network.

The primary reason for defining an application program as privileged is to help ensure network security. Nonprivileged application progams cannot run with the application program name used for a privileged application, even if the privileged application program is not currently running.

Application programs are usually defined as privileged when they are installed in the system. An installed application program is one that resides in the operating system library. The procedure file used to execute an installed application program must have the CASF bit set in the access word associated with the user name in the file. Jobs that maintain or attempt to access installed application programs must also have the CASF bit set in the access words associated with their user names. This bit must be set for access to the system library.

If a privileged application program has not been installed in the system library, it can be executed by a system console operator type-in that invokes its procedure file. The type-in used has the form:

```
X.BEGIN,,anamep.
```

where the anamep parameter is the name of the procedure file. The procedure file must be an indirect access permanent file in the operating system permanent file catalog (stored under the system user name and user index). For the anamep value, you can use a variant of either the program entry point name or the program network application name (NETON statement aname parameter), and all three identifiers should be coordinated. CDC-written application programs are invoked through procedure files for which certain naming conventions have been adopted. These conventions appear in the NOS Installation Handbook, described in the preface. Similar conventions could be adopted for site-written application programs.

An installed privileged application program can be executed by a system console operator type-in of the form:

```
X.anament
```

where the anament parameter is the name of the program (first entry point) installed in the library. For the anament value, you can use a variant of the program network application name (NETON statement aname parameter).

A privileged application program that is not installed can be executed by a system console operator type-in that invokes an installed procedure file. This type-in has the form:

```
X.anamep.
```

where the anamep parameter is the name of the procedure file installed in the library. For the anamep value you can use a variant of either the program entry point or the program network application name (NETON statement aname parameter), and all three identifiers should be coordinated. As described previously, the naming conventions used by CDC for CDC-written application programs should be used as a guide for procedure files invoking site-written application programs.

You should not define an application program as privileged or install it in the system library until the program has been thoroughly debugged. Programs should be run with batch or remote batch origin during the debugging process.

EXECUTION OF APPLICATION PROGRAMS

Application program job structure is partially dependent on the purpose of the job's execution. If the job is executed for debugging purposes, the debugging method chosen for the program can affect the parameters specified in the job's LDSET or LIBRARY command and thereby affect the AIP code executed at the program's control point. This aspect of execution is discussed in the next subsection.

Successful execution of an application program depends on several conditions beyond the scope of the program's code. The less obvious of these dependencies are discussed later in this section; these dependencies are primarily requirements for proper configuration of the program and the terminals it services.

FATAL ERRORS

Portions of the Network Access Method issue diagnostic messages for all fatal errors. These messages are described in appendix B.

The form used for AIP and QTRM diagnostics depends on the library used to load the routines used during execution. When NETIO is used in the LIBRARY or LDSET statement, a single diagnostic message with a reason code is written to the program dayfile before the program is aborted by a fatal error. When NETIOD is used, the same diagnostic is issued, but supplementary diagnostics can also be issued before the program aborts.

DEBUGGING METHODS

Two methods are available for debugging the connection servicing logic of an application program:

Supervisory and/or data message flow through the program can be traced by optional AIP code; this code creates a log file of such messages.

Statistical information on program execution can be gathered for performance adjustment by optional AIP code; this code creates a statistics file of the program's network use.

Debug Log File and Associated Utilities

The optional AIP code that creates the log file gives an application program a means of recording all exchanges between the program and the network. The AIP utility routine NETDBG gives the program a method of selecting exchanges that should be recorded. A running count of the number of messages copied to the debug log file is kept in the supervisory status word (NETON nsup parameter). This count enables the application to decide when to call the AIP utility routine NETREL, which gives an application program a way of releasing, saving, or processing the current information in the debug log file. The AIP utility routine NETSETF gives an application program a way of requesting the operating system to flush the input/output buffer for the debug log file automatically, if the application terminates abnormally. The AIP utility routine NETLOG allows the application to enter messages into the debug log file.

Whether or not the log file is created depends on the system library used to satisfy the application program's externals. AIP code for the program can be loaded from either NETIO or (if the installation elects to install it) from NETIOD. When NETIOD is used, all code needed to create the log file is loaded; the options for logging both supervisory messages and data messages are automatically turned on initially. Because this code causes additional processing overhead and central memory requirements for the application program's control point, you might want to remove the code after the program is completely debugged. You can remove the code from the job without altering the application program's structure by loading the AIP code from NETIO instead of NETIOD. When NETIO is used, the only parts of the log file code loaded are do-nothing versions of NETDBG, NETLOG, NETREL, and NETSETF.

NETDBG Utility

When NETIOD is used, the log file is automatically created without application program calls. You can use calls to NETDBG to switch either or both options for message logging off and back on throughout the program.

NETDBG calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5.) Figure 6-2 shows the NETDBG utility FORTRAN call statement format. NETDBG can only be called after NETON is called and before NETOFF is called.

```
CALL NETDBG(dbugsup, dbugdat, avail)
```

dbugsup — An input parameter that turns the logging of supervisory messages on or off. This parameter can have the values:

- =0 Turn supervisory message logging on.
- ≠0 Turn supervisory message logging off.

When supervisory message logging is turned on, all supervisory messages (except block-delivered messages) exchanged on connection 0 between the application program and NAM are logged. Logging occurs whenever a call to NETGET, NETGETL, NETGETF, NETGTFL, NETPUT, or NETPUTF causes a message transfer. This logging continues until a call with a nonzero debugsup parameter is issued.

dbugdat — An input parameter that turns the logging of data messages on or off. This parameter can have the values:

- =0 Turn data message logging on.
- ≠0 Turn data message logging off.

When data message logging is turned on, all data messages exchanged on any connection between the application program and NAM are logged; block-delivered supervisory messages (FC/ACK/R) are also logged, regardless of the value specified for the dbugsup parameter. Logging occurs whenever a call to NETGET, NETGETL, NETGETF, NETGTFL, NETPUT, or NETPUTF causes a message transfer. This logging continues until a call with a nonzero dbugdat parameter is issued.

avail — A return parameter that indicates whether the logging code portion of AIP was loaded when the program was loaded. On return from the call, this parameter can have the values:

- =0 Loading occurred from NETIOD and logging is possible.
- =1 Loading occurred from NETIO and logging is not possible.

When a value of 1 is returned, specification of 0 for either dbugsup or dbugdat has had no effect but does not cause an error.

Figure 6-2. NETDBG Utility FORTRAN Call Statement Format

Calls to NETDBG can occur in programs using either NETIO or NETIOD. For example, when a NETDBG call turns either or both supervisory and data message logging on and a status is returned indicating logging is not possible, no error occurs and the option selection is ignored. When the program contains a NETDBG call before NETON to turn both logging options off and a status is returned indicating logging is possible, a log file is still created to contain a record of the program's NETON, NETDBG, and NETOFF calls.

NETREL Utility

Log file creation begins when the application program successfully completes its NETON call and ends when NETOFF is issued. If the application has not called NETSETF previously and the program fails, the output buffer used for the log file is not completely emptied into the file. In such a case, the application should reprieve itself and issue a NETOFF call, or a NETREL call, to flush the input/output buffer.

NETREL calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5.) Figure 6-3 shows the NETREL utility FORTRAN call statement format. To use the NETREL utility, an application must issue an initialization call to NETREL with a nonzero first parameter. This call must be issued before NETON and any NETSETF call in order to set up the ZZZZZDN file correctly.

The first parameter on the NETREL call is the name of a file containing a job command record. If the file name supplied does not conform to the NOS operating system file name format, NOS aborts the job when AIP attempts to do input/output on the file.

The second parameter on the NETREL call gives the maximum number of words in each message to be saved in the ZZZZZDN file. NETREL reads up to 192 central memory words of the named file, or until a logical end-of-record is encountered.

The third parameter in the NETREL call determines the position at which NETREL begins reading the named file. The file can be rewound to the beginning-of-information before reading begins, or it can be read from its current position.

After copying the job command record file to the debug log file, AIP writes an end-of-record level 0 to the debug log file before beginning to log messages. Each call to NETREL zeros the message count in the supervisory status word (NETON nsup parameter). Subsequent calls to NETREL route ZZZZZDN to the input queue, reinitialize the file environment table and message count in the supervisory status word, and copy another job command record to a new ZZZZZDN file.

If NETREL is not called and the application is loaded with NETIOD, the debug log file exists as a local file assigned to the application job. The debug log file does not begin with a job command record; therefore, at job termination it should be treated (disposed of) as a normal local file.

```
CALL NETREL(lfn,msglth,nrewind)

lfn       An input parameter that names the
          file containing the job record to be
          copied to the ZZZZZDN file. This
          parameter can have the values:

            =0    The application program job
                  provides its own disposition
                  of the file ZZZZZDN. Only
                  the msglth parameter is proc-
                  essed by AIP.

            ≠0    The named file contains a job
                  record to dispose of the file
                  ZZZZZDN. The value declared
                  for lfn must be left-justified
                  with zero fill, and can be one
                  to seven alphabetic or numeric
                  characters in any combination
                  permitted by the NOS operat-
                  ing system file name format.

msglth    An input parameter that gives the
          maximum number of words of each mes-
          sage to be saved on the ZZZZZDN file;
          0<msglth<410. The value is ignored
          if msglth is 0.

nrewind   An input parameter that controls
          whether AIP rewinds the job record
          file before the NETREL operation
          begins. This parameter can have the
          values:

            =0    File lfn is rewound before
                  any operation is performed.

            ≠0    File lfn is not rewound be-
                  fore any operation is per-
                  formed.

          If the value declared for lfn is zero,
          a value of zero for the rewind para-
          meter is ignored.
```

Figure 6-3. NETREL Utility FORTRAN Call Statement Format

NETSETF Utility

NETSETF calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5.) Figure 6-4 shows the NETSETF utility FORTRAN call statement format. NETSETF allows the input/output buffer for the debug log file ZZZZZDN to be flushed automatically, if the application terminates abnormally. If the error flag code is greater than SPET, then the debug log file is not flushed. See volume 4 of the NOS reference set for a list of the values for the error flag code. Flushing sets the flush bit in the file environment table (FET) for the debug log file and calls the NOS macro SETLOF.

The SETLOF macro provides the NOS operating system with a list of files and FET addresses to be flushed on abnormal termination. The SETLOF macro can be called more than once; each successive call overrides the previous call with a new list of files.

CALL NETSETF(flush,fetadr)

flush — An input parameter that flushes the debug log file automatically upon abnormal termination. The flush parameter can have the following values:

- =0 — the flush bit is set in the FET and the FET address of the debug log file is returned in fetadr.
- ≠0 — the flush bit is set in the FET and the SETLOF macro is called. The FET address is not returned.

fetadr — A return parameter that is the FET address of the debug log file returned by NAM. If zero, either the flush parameter was nonzero or NETIO was loaded (in which case the flush parameter makes no difference).

Figure 6-4. NETSETF Utility FORTRAN Call Statement Format

Applications written in FORTRAN or COBOL should not call NETSETF, because those compilers use CYBER Record Manager, and CYBER Record Manager also calls the NOS macro SETLOF. If you want the application to call the SETLOF macro and include the debug log file in the SETLOF macro list, the application can first call NETSETF to get the FET address of the debug log file. If NETSETF is not called and you want an application to flush the debug log file on abnormal termination, then the program must reprieve itself and call NETOFF or NETREL. NETSETF needs to be called only once and should be called before NETON is called. NETREL does not clear the flush bit in the FET when it reinitializes the FET.

NETLOG Utility

NETLOG calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5). Figure 6-5 shows the NETLOG utility FORTRAN call statement format. NETLOG allows an application to enter messages into the debug log file. These messages can be of any size, but large messages degrade the performance of AIP. Messages are copied to the debug log file unchanged. However, they are truncated if the NETREL utility has previously been called and if the message length exceeds the number of central memory words specified as the maximum message length in the NETREL call. The messages can be either formatted or unformatted. DLFP prints formatted messages unchanged. DLFP prints unformatted messages the same way it prints network message text (in octal, hexadecimal, display code, and ASCII characters). NETLOG cannot be called before NETON.

NETDMB Utility

NETDMB calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5). Figure 6-6 shows the NETDMB utility FORTRAN call statement format. NETDMB allows an application to dump its exchange package and central memory field

CALL NETLOG(address,size,format)

address — An input parameter that gives the address of the message to be written to the debug log file.

size — An input parameter that gives the size in central memory words of the message to be written to the debug log file.

format — An input parameter that determines whether the message is formatted or unformatted. This parameter can have the values:

- =0 — The message is unformatted and will be printed by DLFP in octal, hexadecimal, 6-bit display code characters, and ASCII characters.
- =1 — The message is formatted and will be printed unchanged by DLFP.

Figure 6-5. NETLOG Utility FORTRAN Call Statement Format

CALL NETDMB(dumpid,ecs)

dumpid — An input parameter that is a six-digit octal number dump identifier. $0 \leq dumpid \leq 777777$

ecs — An input parameter that, if ≠0, also dumps the associated ECS.

Figure 6-6. NETDMB Utility FORTRAN Call Statement Format

length into the local dump file ZZZZDMB. The data is in binary format. The file ZZZZDMB must be postprocessed by a binary dump interpreter to allow selection of address range and formatting for print. The dump formatting is done through DSDI which is described in the NOS 2 System Maintenance reference manual. A logical end-of-record is written to the file ZZZZDMB after each NETDMB call.

Debug Log File Postprocessor Utility

The debug log file is a binary compressed file and it is written using NOS data transfer macros. You can obtain a listing of this file by running the debug log file postprocessor utility with the desired options.

The debug log file postprocessor (DLFP) utility is a program that processes the debug log file generated by AIP. The general format of the DLFP control statement is shown in figure 6-7. Examples of DLFP control statements are shown in figure 6-8.

The job command format for DLFP is:

DLFP(p_1,p_2,p_3,p_4,p_5)

p_i is any of the following parameters in any order:

I=lfn_1	Directives comprise the next record on file lfn_1.
I=0	No directive input.
I omitted	Directives on file INPUT.
L=lfn_2	List output on file lfn_2.
L omitted	List output on file OUTPUT.
B=lfn_3	File lfn_3 contains the debug log file.
B omitted	Debug log file is ZZZZZDN.
D	Discontinue processing current directive record if there are errors in it. Restart with next directive record if any.
D omitted	Do not ignore directive errors; abort job.
N=lfn_4	Create new debug log file lfn_4 with records selected from lfn_3 or ZZZZZDN according to directives governing record selection for the list output file. If this option is selected, no debug log file data is written on the list output file.
N omitted	No new debug log file is created.

File names must comply with the NOS product set format.

Figure 6-7. DLFP Control Statement General Format

DLFP(I=0,B=TAPE)	DLFP reads the debug log data from file TAPE. The entire log file is processed and written to output. The output goes to the OUTPUT file.
DLFP(D,L=SAVE)	DLFP reads the debug log data from file ZZZZZDN. DLFP reads the INPUT file looking for directives. If the directives are not correct, DLFP ignores them. The output goes to file SAVE.
DLFP(I=DIR,B)	DLFP aborts with the fatal error message PARAMETER FORMAT ERROR because there is no file associated with the B parameter. If the B parameter is specified correctly, DLFP reads file DIR looking for directives. If the directives are not correct, DLFP aborts.

Figure 6-8. DLFP Job Command Examples

The debug log file postprocessor automatically rewinds the debug log file before postprocessing begins. The application programmer needs to rewind the file only when DLFP is not the first software to access the file after program execution completes.

The debug log file can be copied, made permanent, or otherwise accessed before DLFP begins its postprocessing. Such operations, however, must not alter the form of the file used for DLFP input. You cannot copy portions of the file and successfully run DLFP using the incomplete copy.

The N option of the DLFP command provides a means for creating a new debug log file that is a subset of an existing debug log file. The new file can be separately processed by a subsequent DLFP job command and separate DLFP directives.

An optional directive file can be submitted to the DLFP to select special messages for output. The directive file can have zero or more directive records.

Each directive record is a Z type record, which can contain one or more keywords starting in card image column 1. Keywords allow you to select which messages are written to the output file. All keywords are optional and can appear in any order. You can use one or more blanks, or a comma followed by zero or more blanks, to separate the keywords. You can use leading blanks. Figure 6-9 shows the general format of DLFP directive keywords with examples of them in figure 6-10.

Each directive record initiates an independent search. An empty directive file or empty directive record or I=0 causes all debug log file messages to be output. Directive records are copied to the output listing file.

DLFP issues dayfile messages to inform users of fatal errors or processing completion. Appendix B provides a list of all dayfile messages issued by DLFP. Errors or informative messages can be written to the output file by DLFP. All messages except NO MESSAGES FOUND are fatal errors and cause the job to be aborted unless the D option was specified on the job command.

The general format of a log file listing is shown in figure 6-11. (Appendix I includes a sample output.) NETON and NETOFF calls are logged to record the start and end of NAM interfacing; only successful NETON calls are logged. Each AIP call logged is followed by the octal relative address (in parentheses) from which the call was made. The NETON call log includes the parameter values declared on the statement. The NETDBG call log lists the value declared for dbugsup as OPT1 and for dbugdat as OPT2. Calls that transfer messages are logged with their declared parameters, followed by the block header contents and data message contents. (All words comprising a supervisory message are listed.) The contents of each word are given in hexadecimal, octal, 6-bit display code form, and ASCII-coded form. Each message is numbered in the order it was transferred.

Keyword†	Value	Description
B		Specifies that only upline blocks with the flow control break flag bit (bit brk) set in the application block header are output.
BD=	yymmdd	Specifies that only messages that were logged on or after this date are output. Messages before this date are not output. yy is the rightmost two digits of the year, mm is the month, and dd is the day of the month; $00 \leq yy \leq 99$, $01 \leq mm \leq 12$, $01 \leq dd \leq 31$.
BT=	hhmmss	Specifies that only messages that were logged on or after this time are output. Messages before this time are not output. If the debug log file contains more than one day's messages, messages beginning after the first occurrence of this time will be output if BD is not specified. hh is the hour, mm is the minute, and ss is the second; $00 \leq hh \leq 24$, $00 \leq mm \leq 59$, $00 \leq ss \leq 59$.
C		Specifies that only messages with the Cancel flag set in the application block header are output.
CN=	n	Specifies that only synchronous and asynchronous supervisory messages and data blocks relating to connection number n are output; $1 \leq n \leq 255$.
DN=		Reserved for CDC use.
E		Specifies that only messages with the error bit set in the supervisory message are output.
ED=	yymmdd	Specifies that messages on or after this date are not to be output. yy is the rightmost two digits of the year, mm is the month, and dd is the day of the month; $00 \leq yy \leq 99$, $01 \leq mm \leq 12$, $01 \leq dd \leq 31$.
ET=	hhmmss	Specifies that messages on or after this time are not to be output. If the debug log file contains more than one day's messages, searching terminates after the first occurrence of this time if ED is not specified. hh is the hour, mm is the minute, and ss is the second; $00 \leq hh \leq 24$, $00 \leq mm \leq 59$, $00 \leq ss \leq 59$.
LE=	n	Specifies maximum length in central memory words of each message to be output; $1 \leq n \leq 410$ (default=10).
F		Specifies that only network messages with the no format effector bit set in the application block are output.
N		Specifies that only network messages are output. Messages generated by applications for the debug log file are ignored.
NM=	n	Specifies that only n messages are output; $0 \leq 1000000$.
P=		Specifies that only messages with the parity-error bit or auto input mode bit set in the application block header are output.
PF=	hh	Specifies that only supervisory messages with the primary function code (PFC) equal to hh_{16} are output. No check is made to determine whether hh is a legal PFC value; $00 \leq hh_{16} \leq FF$.
PS=	hhxx	Specifies that only supervisory messages with PFC/SFC equal to $hhxx_{16}$ are output. No check is made to determine whether hh is a legal PFC value and xx is a legal SFC value. $0000 \leq hh_{16} \leq FFFF$.
R		Specifies that only messages with the response bit set in a supervisory message are output.
SM=	n	Specifies that no messages are output until after the nth message, which satisfies all the other keyword options, is found; $0 \leq n \leq 1000000$.
SN=		Reserved for CDC use.
T		Specifies that only upline blocks with the data truncation flag bit set in the application block header are output.

Figure 6-9. DLFP Directive Keyword Format (Sheet 1 of 2)

Keyword†	Value	Description
U		Specifies that only messages with the input block undeliverable bit set in the application block header are output.
X		Specifies that only messages with the transparent data bit set in the application block header are output.

†The same keyword can appear more than once in a directive record. If there is a value associated with this keyword, the value in the last occurrence of the keyword is the one used for the search. Blanks can precede or follow the = sign. If both PF and PS are specified, the last one specified overrides the first one specified. If there are errors in the directive record, the job is aborted unless the D option was specified on the control statement. If the D option was specified, the directive record in error is ignored and processing restarts with the next directive record, if any. If there are multiple errors in a directive record, all errors are identified.

Figure 6-9. DLFP Directive Keyword Format (Sheet 2 of 2)

R,E	DLFP processes and outputs all supervisory messages that have both the response and error bit set. There are currently no supervisory messages that have both bits set.
BD=780229,BT=2401,ED=780228	DLFP does not process this directive record because it contains at least one error. The first error is that February 29, 1978 is an invalid date. The second error is that 2401 is an invalid time. Note that it was not an error to have the ED date earlier than the BD date although no messages would ever be processed because of it.
PF=ABC,SM=-1,LE=1F,NM=10000000	DLFP does not process this directive record because it contains at least one error. The first error is that ABC is not a two-character hexadecimal number. The second error is that - is not a legal character to have in the directive record. The third error is that 1F is not a decimal number. The fourth error is that the character string NM=10000000 is greater than 10 characters.
X,CN=15,SM=20	DLFP processes and outputs all messages on connection number 15 that have the transparent bit set except for the first 19.
PS=8301,CN=5,PF=83	DLFP processes and outputs all supervisory messages relating to connection number 5 that have a PFC=83_{16}(FC mnemonic). Note that even though PS is also specified, the directive is ignored because PF is specified after it.
BC=781104,BT=2350,ED=781105, ET=000000	DLFP processes and outputs all messages that occurred from 11:50 PM on November 4, 1978 to midnight.
LE=2,PF=67,NM=10	DLFP processes the first ten supervisory messages with PFC=67_{16}(CON mnemonic). Only the first two words of each supervisory message is output.
PS=8381	DLFP outputs no messages. 81 is too large a value for SFC, so DLFP does not find any matching supervisory message.
PS=6302,CN=1,E	DLFP processes and outputs all CON/ACRQ/R supervisory messages relating to connection number 1 that have the error bit set.
,CN=300,UX,PF=FD,CN=30	DLFP does not process this directive record because it contains at least one error. The first error is that the first keyword does not begin in column 1. The second error is that 300 is too large a connection number. The third error is that there should be a comma or blank between the U and X. Even if the three errors were not present, DLFP would not output any messages because currently FD is not a legitimate PFC value. Also CN=30 does not fix the error in the first CN directive.

Figure 6-10. DLFP Directive Examples

```
                                    aname LOG FILE OUTPUT                          current date   yy/mm/dd
                                    DATE RECORDED yy/mm/dd                                        PAGE ddd


hh.mm.ss.mil NETON (oooooo) ANAME = ccccccc DATE = yy/mm/dd                                       MSG NO. ddd
    NSUP ADDR = oooooo MINACN = dddd MAXACN = dddd

hh.mm.ss.mil NETDBG (oooooo) OPT1 = b OPT2 = b DATE = yy/mm/dd                                    MSG NO. ddd

hh.mm.ss.mil NETGET (oooooo) ACN = dddd HA = oooooo TA = oooooo TLMAX = dddd                      MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
                                          .
                                          .
        nnn   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa

hh.mm.ss.mil NETLOG (oooooo)                                                                      MSG NO. ddd

        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
        003   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa

hh.mm.ss.mil NETGETL (oooooo) ALN = dddd HA = oooooo TA = oooooo TLMAX = dddd                     MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
                                          .
                                          .
        nnn   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa

hh.mm.ss.mil NETGETF (oooooo) ACN = dddd HA = oooooo NA = dd TAA = oooooo                         MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

    FRAGMENT    1    SIZE = dddd  ADDRESS = oooooo
        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
    FRAGMENT    2    SIZE = dddd  ADDRESS = oooooo
                                          .
                                          .
    FRAGMENT    dd   SIZE = dddd  ADDRESS = oooooo
        nnn   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa

hh.mm.ss.mil NETGTFL (oooooo) ALN = dddd HA = oooooo NA = dd TAA = oooooo                         MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

    FRAGMENT    1    SIZE = dddd  ADDRESS = oooooo
        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
                                          .
                                          .
    FRAGMENT    dd   SIZE = dddd  ADDRESS = oooooo
        nnn   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa

hh.mm.ss.mil NETPUT (oooooo) HA = oooooo TA = oooooo                                              MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

        001   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
                                          .
                                          .
        nnn   hhhhhhhhhhhhhhh     ooooooooooooooooooo      ccccccccccc aaaaaaaaa
```

Figure 6-11. General Format of DLFP Output (Sheet 1 of 2)

```
hh.mm.ss.mil NETPUTF (oooooo) HA = oooooo NA = dd TAA = oooooo                    MSG NO. ddd
    ABT = dd ADR = dddd ABN = oooooo ACT = dd STATUS = bbbbbbbb TLC = ddd

    FRAGMENT    1    SIZE = dddd  ADDRESS = oooooo
        001   hhhhhhhhhhhhhhh     oooooooooooooooooo      cccccccccc aaaaaaaaa        mnemonic
                                        .
                                        .
        nnn   hhhhhhhhhhhhhhh     oooooooooooooooooo      cccccccccc aaaaaaaaa

    FRAGMENT    dd   SIZE = dddd  ADDRESS = oooooo
        nnn   hhhhhhhhhhhhhhh     oooooooooooooooooo      cccccccccc aaaaaaaaa

hh.mm.ss.mil NETOFF (oooooo) DATE = yy/mm/dd.                                    MSG NO. ddd
```

LEGEND:

aname	Application name.
hh.mm.ss.mil	System clock time of the AIP call in hours, minutes, seconds, and milliseconds.
yy/mm/dd	System date expressed as year, month, and day.
mnemonic	For supervisory messages, the message mnemonic appears; for data messages, this area is blank.
a ... a	Indicates ASCII characters are listed.
b ... b	Indicates binary digits are listed.
c ... c	Indicates display code characters are listed.
d ... d	Indicates decimal digits are listed.
h ... h	Indicates hexadecimal digits are listed.
o ... o	Indicates octal digits are listed.
n ... n	Indicates last central memory word listed from block.

Figure 6-11. General Format of DLFP Output (Sheet 2 of 2)

The listing provides the following labeled information:

ACN gives the value used for the acn parameter in the indicated call.

ALN gives the value used for the aln parameter in the indicated call.

HA gives the octal relative address used in place of the symbolic address specified for the ha parameter in the indicated call.

TA gives the relative address used in place of the symbolic address specified for the ta parameter in the indicated call.

NA gives the value used for the na parameter in the indicated call.

TAA gives the relative address used in place of the symbolic address specified for the taa parameter in the indicated call.

TLMAX gives the value used for the tlmax parameter in the indicated call.

ABT gives the abt field content for the application block header used in the indicated call.

ADR gives the adr or acn field content for the application block header used in the indicated call.

ABN gives the abn field content for the application block header used in the indicated call.

ACT gives the act field content for the application block header used in the indicated call.

STATUS gives the settings of bits 19 through 12 for the application block header used in the indicated call, at the time the call is completed.

TLC gives the tlc field content for the application block header used in the indicated call.

FRAGMENT gives the number within the call taa array used to locate the corresponding information transferred by the call.

SIZE gives the content of the size field within the call taa array used to delimit the corresponding information transferred by the call.

ADDRESS gives the address field content of the taa array used to locate the corresponding information transferred by the call.

Statistical File and Associated Utilities

The optional AIP code that creates the statistical file allows you to record cumulative figures of exchanges between the program and the network. The AIP utility routine NETSTC gives the program a method of selecting which portions of the program have figures accumulated. The AIP utility NETLGS allows you to write messages in the statistical file. All statistical output is written to a local file named ZZZZZSN.

Whether or not the statistical file is created depends on the system library used to satisfy the application program's externals. AIP code for the program can be loaded from either NETIO or (if the installation elects to install it) from NETIOD. When NETIOD is used, all code needed to create the statistical file is loaded; accumulation of figures is automatically turned on initially. Because this code causes additional processing overhead and central memory requirements for the application program's control point, you can remove the code when the statistical file is not needed. You can remove the code from the job without altering the application program's structure by loading the AIP code from NETIO instead of NETIOD. When NETIO is used, the only part of the statistical file code loaded is a do-nothing version of NETSTC.

When NETIOD is used, the statistical file is automatically created without application program calls. You can use calls to NETSTC to switch accumulation off and back on throughout the program, and to dump and restart statistics counters.

NETSTC Utility

NETSTC calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5.) Figure 6-12 shows the NETSTC utility FORTRAN call statement.

Calls to NETSTC can occur in programs using either NETIO or NETIOD. For example, when a NETSTC call turns accumulation on and a status is returned indicating accumulation is not possible, no error occurs and the option selection is ignored. When the program contains a NETSTC call immediately after NETON to turn accumulation off and a status is returned indicating accumulation is possible, a statistical file is still created to contain a record of the program's NETON, NETSTC, and NETOFF calls. A call to NETSTC before NETON is legal.

Statistical file creation begins when the application program successfully completes its NETON call and ends when NETOFF is issued. A logical end-of-record is written to file ZZZZZSN when NETOFF is called. Because the output buffer used for the file is not completely emptied into the statistical file until the application program issues a NETOFF call, it is important to issue the call even when the program loses communication with the network; otherwise, the last few entries written to the statistical file for the job run cannot be saved. All statistics are written to file ZZZZZSN and the counters reset to zero whenever a call to NETSTC is made to turn statistics gathering off and AIP was loaded from NETIOD. Individual statistics are written to ZZZZZSN and reset to zero whenever the counter overflows.

NETLGS Utility

NETLGS calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5). Figure 6-13 shows the NETLGS utility FORTRAN call statement format. NETLGS allows an application to enter messages into the statistical log file ZZZZZSN.

CALL NETSTC(onoff, avail)

onoff — An input parameter that turns the accumulation of statistics on or off. This parameter can have the values:

- =0 Turn accumulation on.
- =1 Turn accumulation off.

When statistics accumulation is turned on, each call to an AIP routine increments a counter for that routine and each block transferred between the application program and the network increments a counter for blocks of that type. Incrementing continues until a call with an onoff parameter of 1 is issued. Calls with onoff parameters of 0 cause the counters to be reset to 0.

avail — A return parameter that indicates whether the statistics accumulation portion of AIP was loaded when the program was loaded. On return from the call, this parameter can have the values:

- =0 Loading occurred from NETIOD and accumulation is possible.
- =1 Loading occurred from NETIO and accumulation is not possible.

When a value of 1 is returned, specification of 0 for the onoff parameter has no effect but does not cause an error.

Figure 6-12. NETSTC Utility FORTRAN Call Statement Format

```
CALL NETLGS(address,size)

address     An input parameter that indicates the
            address of the message to be written
            to the statistics log file. The
            message must contain 6-bit display
            code information with a line termi-
            nator (12 to 66 bits of zero, right-
            justified in a central memory word).

size        An input parameter that indicates the
            number of words in the message.
```

Figure 6-13. NETLGS Utility FORTRAN Call Statement Format

When application program execution ends, the statistical file exists as a local file named ZZZZZSN. The file is written using NOS data transfer macros; the contents are 6-bit display code characters, formatted for printer output. To obtain a listing of this file, the file must be rewound and copied to OUTPUT, or otherwise disposed by using ROUTE.

Each period for which statistics are accumulated during program execution is listed separately in the statistical file. Figure 6-14 shows the general format of the period listings. The counters used are 60-bit signed integers, reset to zero at the beginning of each period. If a counter is not used during a given period (its value remains zero), the corresponding line for the counter is omitted from the listing for that period. If a counter overflows during a given period, the corresponding line in the listing is preceded by the message:

```
****COUNTER OVERFLOW****
```

and the counter is reset to zero. If the program is running in parallel mode during the period, the number of transfer attempts unsuccessful because NIP was busy are listed. The CPU utilization shown is cumulative between the NETON and NETOFF calls. The NAK-S line indicates the number of block-not-delivered (FC/NAK/R) supervisory messages received.

DEPENDENCIES

Before an application program executes as part of the network, the application program should be identified in the network's files as part of the local host computer system's resources. This is done by entering its application program name into the local configuration file, using the Network Definition Language (NDL). This action is not the application programmer's responsibility and is not described in this manual. Use of the Network Definition Language is described in the Network Definition Language reference manual mentioned in the preface.

```
NAM STATISTICS GATHERING STARTED

NET {ON } DATE yy/mm/dd. TIME hh.mm.ss.
    {STC}

NAM STATISTICS GATHERING TERMINATED

NET {OFF} DATE yy/mm/dd. TIME hh.mm.ss.
    {STC}

CPU TIME USED:   dddddd SEC

NUMBER OF PROCEDURE CALLS

    NETCHEK      dddddd
    NETGET       dddddd
    NETGETF      dddddd
    NETGETL      dddddd
    NETGTFL      dddddd
    NETPUT       dddddd
    NETPUTF      dddddd
    NETSETP      dddddd
    NETWAIT      dddddd

NUMBER OF WORKLIST TRANSFER ATTEMPTS

    SUCCESSFUL       dddddd
    UNSUCCESSFUL     dddddd

NUMBER OF INPUT/OUTPUT BLOCKS TRANSFERRED

    INPUT     ABT=0  dddddd
    INPUT     ABT=1  dddddd
    INPUT     ABT=2  dddddd
    INPUT     ABT=3  dddddd
    OUTPUT    ABT=1  dddddd
    OUTPUT    ABT=2  dddddd
    OUTPUT    ABT=3  dddddd

NUMBER OF ERRORS

    LOGICAL ERROR    dddddd
        NAK-S        dddddd

Legend:

    yy/mm/dd      System date of the call beginning
                  or ending the accumulation period,
                  expressed as year, month, and day

    hh.mm.ss      System clock time of the call
                  beginning or ending the accumu-
                  lation period, expressed in hours,
                  minutes, and seconds

    d. . . d      Indicates decimal digits
```

Figure 6-14. General Format of One Period Listing in Statistical File

Until the application program is identified in the NOS system COMTNAP common deck, the program cannot call NETON and execute with actual logical connections made. Until configured as a network resource, the program's connection-servicing logic cannot be debugged.

When the program is identified in COMTNAP, it can successfully perform a NETON call if the network is operational. As soon as a NETON call is completed, terminals can request connection to the program.

Before a terminal can complete a connection to the program, the user name from its login procedure must have an access word bit associated with the application program's name in COMTNAP. This association is established by using MODVAL and must exist for all login user names. The procedure is not described further in this manual because it is not the application programmer's responsibility.

If the application program uses the batch device interface, the owning console for the passive device it is intended to service must be configured in the local configuration file with the program declared as the initial application for the terminal. Unless this is done, the passive devices cannot access the application program. The application programs released by CDC with this version of the network software only provide a mechanism for the switching of interactive device connections to other programs. A passive device configured with the Remote Batch Facility as its initial application program cannot be used by any other application program.

MEMORY REQUIREMENTS

Although the size of an application program varies with its complexity and functions, the AIP coding added by the CYBER loader does not normally exceed 1100 words of central memory. The version of AIP that generates the debug log file and statistics file requires 1100 more words. Using the QTRM utility package adds less than 700 additional words to the program's central memory field length requirements.

THE COMMUNICATIONS CONTROL PROGRAM AND THE NETWORK PROCESSING UNIT

This section contains a general description of how the Communications Control Program (CCP) functions in a network processing unit (NPU). This description:

- Introduces the major hardware components of the NPU that form the hardware environment for CCP operation.
- Briefly describes the multiplex subsystem and the mechanics of transferring data in the two general formats used by the network; these are the interactive virtual terminal (IVT) format defined in section 2, and the physical record unit (PRU) format used to support CDC-defined batch devices through the special batch data interface mentioned in section 2.
- Summarizes the operation of CDC-written terminal interface programs (TIPs).

HARDWARE ENVIRONMENT

Each CDC network processing unit consists of:

- A 2551 series Communications Processor, with macromemory and micromemory
- A multiplex subsystem, containing varying numbers and types of communications line adapters
- 0, 1, or 2 CYBER channel couplers

Figure 7-1 shows the maximum architecture of such a network processing unit. Some NPUs also have a magnetic tape cassette drive and an operator's console that is not part of the network.

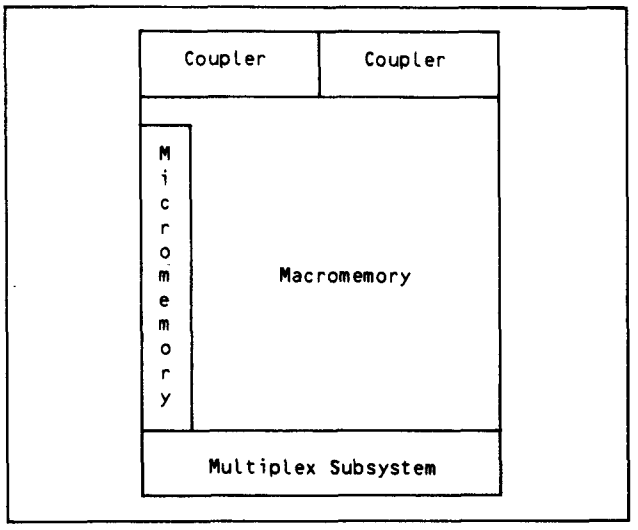


Figure 7-1. Basic Components of a CDC Network Processing Unit

2551 SERIES COMMUNICATIONS PROCESSOR

The 2551 series Communications Processor has both a programmable micromemory and a macromemory; each model of the 2551 series has varying amounts of macromemory, with several sizes of random access micromemory (RAM) logic. 8192 words (8K) of micromemory are required to run the current release of CCP.

Because models can be modified on site, a model designation does not necessarily correspond to a specific macromemory or micromemory configuration. The internal characteristics of a specific 2551 Communications Processor model are described in the corresponding hardware reference manual listed in the preface.

Each NPU might require its own variant of CCP, depending on the macromemory size of the unit and the software modules that need to reside in the unit. Variants are generated as described in the NOS Version 2 System Maintenance Reference Manual described in the preface. These variants reside in the host NPU load file (NLF). The appropriate variant for loading into each NPU is determined by the host Network Supervisor program from information contained in the network configuration file. This process is described in the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface.

CYBER CHANNEL COUPLER

Local NPUs (front-end NPUs for CDC host computers) have at least one CYBER channel coupler. Although up to two channel couplers can be physically housed in an NPU cabinet, the channel coupler is treated as an external device by the NPU. The coupler contains programmable registers; it serves as a transfer mechanism to convert the 8-bit data bytes within one 16-bit NPU memory word into 8-bit data bytes within two 12-bit words of a CDC CYBER 170 computer's peripheral processor.

CCP services the coupler through a software module called the Host Interface Program (HIP). The external characteristics of the CYBER channel coupler are described in the Network Access Method Version 1 Network Definition Language Reference Manual mentioned in the preface. The internal characteristics of a coupler are described in the appropriate 2551 Communications Processor model hardware reference manual listed in the preface.

CASSETTE DRIVE

The magnetic tape cassette drive and an accompanying deadman timer hardware is required if the 2551 has two couplers or is used as a remote NPU. When an NPU has a cassette drive, the drive is used to begin the loading of CCP from a host computer.

The cassette drive loads a copy of the system autostart module program (SAM-P) from a CDC-supplied cassette. SAM-P is essentially a bootstrap loader that obtains the copy of CCP appropriate for its NPU from a copy of the Network Supervisor program in a host computer.

The cassette drive can also be used to load optional offline hardware diagnostic software for use with the offnet NPU console.

NPU CONSOLE

If the NPU has an operator's console, that console is not part of the network; it connects to a special port of the NPU and is not serviced through a communications line adapter. This offnet console is not the console associated with the NPU operator who is known to the Communications Supervisor program in the host as a network operator or NOP.

The offnet NPU console is used to run optional online or offline diagnostic software; use of that software is described in the Communications Control Program Version 3 Diagnostic Handbook listed in the preface. If the site modifies its copy of CCP, the offnet console can be used with the internal Test Utility Package for online debugging of the code.

MULTIPLEX SUBSYSTEM

The multiplex subsystem contains the following components:

A multiplex loop interface adapter (MLIA)

One or more loop multiplexers (LMs)

One or more communications line adapters (CLAs)

Related firmware (microprograms) and software

The basic relationships of these components are shown in figure 7-2.

There are three types of CDC communications line adapters:

Model 2560 series synchronous CLAs, used to support lines connecting Mode 4 protocol, IBM 2780 or IBM 3780 bisynchronous protocol, or HASP protocol terminals to the NPU

Model 2561 series asynchronous CLAs, used to support lines connecting teletypewriter-compatible or IBM 2741-compatible terminals to the NPU

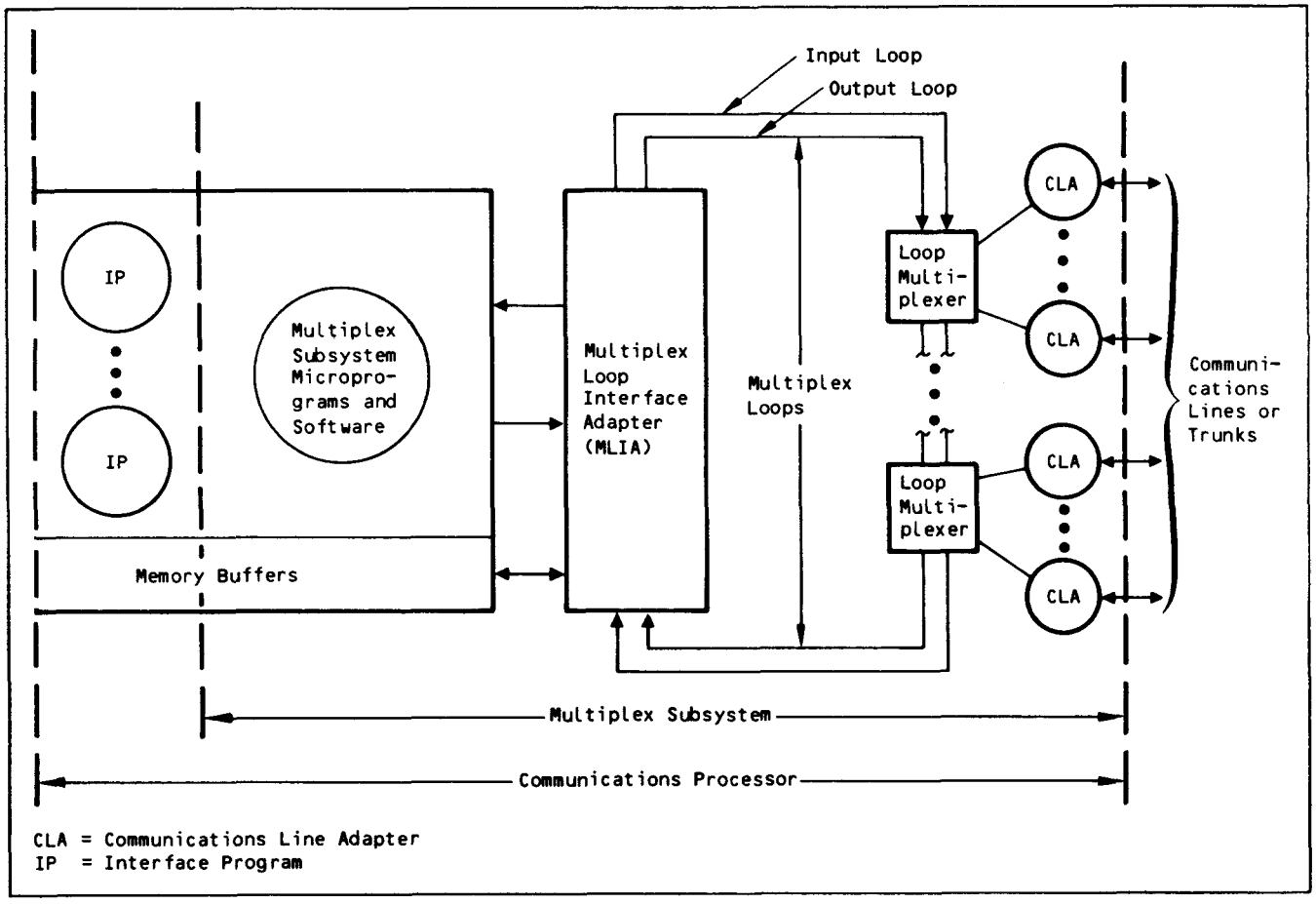


Figure 7-2. Basic Components of the Multiplex Subsystem

Model 2563 series high-level data link control (HDLC) CLAs, used to support trunks connecting other NPUs or lines connecting asynchronous terminals to the NPU through X.25 packet-switching networks

The types of CLAs in the multiplex subsystem of a specific NPU depend on the types of connections that NPU must support. The external characteristics of CLAs are described in the Network Access Method Version 1 Network Definition Language Reference Manual mentioned in the preface. The internal characteristics and functions of CDC-supplied CLAs are described in their respective hardware reference manuals listed in the preface.

MULTIPLEX SUBSYSTEM OPERATION

The following paragraphs provide a brief overview of the flow of data through all of the hardware and software components listed above.

The multiplex subsystem has two principal tasks:

Relieves the TIPs of having to process communication line data according to physical line characteristics (most line characteristics are invisible to the TIPs as a result of multiplex subsystem processing)

Multiplexes data to match the low-speed characteristics of individual terminals with the high-speed characteristics of the NPU and the host

The multiplex subsystem:

Receives serial data from communications lines, places it in a circular input buffer, demultiplexes it, and places it into a separate input buffer for each device on a communication line

Detects and processes special characters

Translates code on input

Detects breaks in communication

Checks the cyclic redundancy count (CRC)

Checks and generates character parity

Controls modems on the NPU end of communication lines and analyzes line status

Processes commands issued by the TIPs

Chains input data buffers as necessary to assemble network data blocks

Transmits serial data to communications lines from communications line output buffers

INPUT MULTIPLEXING

Each line has a communications line adapter (CLA). If a character is ready when each CLA is sampled, the CLA places the character on the input multiplex loop together with information identifying the source (line) and in some cases, the nature of the character (for instance, a character byte can contain modem signal information rather than a character of data). All information on the input multiplex loop is routed to a circular input buffer (CIB) which is usually 512 words long. The demultiplexing operation picks data from this buffer and reconstitutes the data input lines separately for each communications line.

INPUT DEMULTIPLEXING

All TIPs except the HASP and BSC TIPs and possibly a site-written TIP process data in a single pass. For one-pass TIPs, the multiplex subsystem is responsible for selecting data from the CIB as well as for putting it into a communication line input buffer.

When transmission block input starts, the multiplex subsystem reserves a data buffer for the network block (data buffers are normally 64 words long, with 2 characters per NPU word). The words in the CIB are identified by the communication line number, and are packed into an input buffer for that line. If a buffer is filled before the transmission block is complete, another buffer is assigned and is chained to the first.

When the state programs detect the end of input, the TIP appropriate for the terminal type is notified. The TIP continues the processing by passing control of the data to the BIP, which formats the data into network blocks and controls the routing of those blocks. The BIP passes control of the network blocks to the Host Interface Package (HIP) or the Link Interface Package (LIP).

The LIP passes the network block across a trunk communications line to another LIP, as described below under Trunk Multiplexing. The HIP passes the network block through the coupler to the PPU of the host.

OUTPUT MULTIPLEXING

When the NPU has received a network data block from the host and a TIP has transformed the code and/or format from IVT or PRU to terminal format, the TIP notifies the multiplex subsystem that the transmission block is ready for output. The multiplex subsystem picks the characters from the communication line's output buffer one at a time, whenever a new output data demand (ODD) is generated by the CLA. (The ODD indicates the communications line is ready to receive another character.)

The outgoing characters are placed on the output multiplex loop, along with such control characters as are needed, and an address that will be recognized by the CLA for the active line to that terminal device. The CLA recognizes the address, picks that data from the output loop, and marks it as being sent. When the contents of the entire output buffer for the communications line have been transmitted, and the transmission block has been received by the terminal, the multiplex subsystem can notify the TIP (or the TIP can wait for an acknowledgment response from the terminal). The TIP releases the output buffer and notifies the host of a successful transmission.

OUTPUT DEMULTIPLEXING

The demultiplexing of downline transfers is a terminal function; that is, the network block is reconstituted in a buffer for the terminal. That buffer becomes a transmission block to the terminal, which must then subdivide or route the block for delivery to the proper output device.

TRUNK MULTIPLEXING

If a remote NPU is included in the network, transmissions between the local NPU and the remote NPU take place over a trunk communications line. In the local NPU, a link interface package (LIP) sets up the output buffer, and the network block in that buffer remains in IVT or PRU format while being transmitted over the trunk. In the remote NPU, the downline network block goes through the CIB and is then demultiplexed by the LIP.

The LIP passes the block to the BIP, which performs the operations described under Input Multiplexing. The BIP then passes the block to the appropriate TIP.

After the TIP in the remote NPU translates the code from IVT or PRU format to terminal format, the network block is treated as an output network block in a local NPU; that is, the block is multiplexed for transmission, is sent, and is acknowledged. The acknowledgment must be formatted as any other input network block: remultiplexed and sent upline through the local NPU to the host.

Input network blocks (upline traffic) are reformatted from terminal to PRU or IVT format as in a local NPU, but are then sent by the LIP through the remote NPU multiplexer, received by the local NPU multiplexer, and reconstructed into complete network blocks in the local NPU by that NPU's LIP. Network blocks passing through two NPUs are multiplexed twice in a remote NPU, once to/from the terminal and again from/to the local NPU.

NETWORK BLOCK HANDLING

The basic procedures for upline and downline network block movement are discussed next. Note that the procedures given are highly summarized. Acknowledgment procedures are also highly summarized. It is assumed that terminals are connected through a single NPU.

SIMPLIFIED INPUT PROCESSING

Figure 7-3 shows the movement of data from a terminal to a host applications program. Solid lines indicate data and acknowledgment pathways, dashed lines indicate principal control functions.

The major steps in upline data processing are:

Polling synchronous terminals (such as Mode 4 terminals) to find if the terminal has data ready to send.

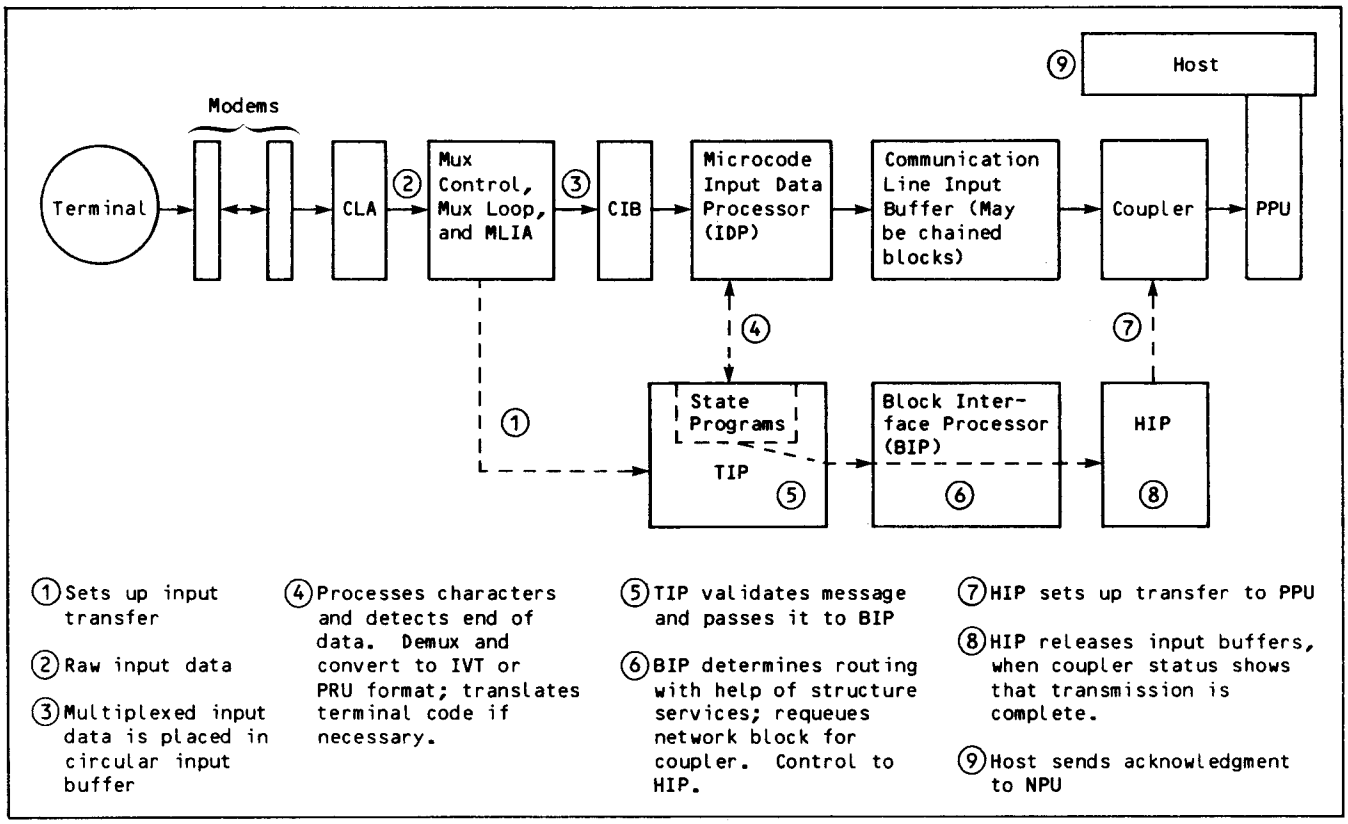


Figure 7-3. Simplified Input Processing

Setting up the multiplex subsystem and buffers when a terminal indicates it has data to send upline.

Collecting data from all active terminals in a circular input buffer (CIB).

Demultiplexing data and converting it to IVT or PRU format, a process controlled by state programs unique to each TIP. Demultiplexed data is collected in one or more chained buffers that are collectively called a communication line input buffer. If code conversion is necessary (such as EBCDIC to ASCII for IVT data or EBCDIC to display code for PRU data), this is also accomplished.

When an input buffer is full (that is, the transmission block is complete), the block content is validated for conformance to the appropriate protocols.

The transmission block is converted to one or more network blocks.

Network block routing is determined and the block is queued to the host coupler.

Line statistics (a type of status information that includes both success and failure data, such as number of blocks and characters processed or the number of transmission errors detected) are updated for the transfer. These statistics are periodically sent upline to the host; some become accounting messages, others become hardware error log messages.

The coupler transmits the block to the host PPU and dequeues the block.

The host receives the block and queues it to the applications program.

The NPU releases the block buffer.

When the host delivers the block to an application program, it sends an acknowledgment to CCP.

SIMPLIFIED OUTPUT PROCESSING

Figure 7-4 shows the movement of data from a host application program to an interactive terminal (downline blocks). Conventions for solid and dashed lines are the same as for the input diagram.

The text below describes the same movement for data sent to a CDC-defined batch device; the same steps occur for both device types, but the order of operations varies. The major steps in downline data processing are:

An interrupt from the host indicates a buffer of data is ready for transmission. The data is in PRU or IVT format at this point.

If the NPU is not already saturated with other, higher priority tasks (note that output takes precedence over input), the host interface package (HIP) sets up the coupler to receive the network block and assigns a buffer (or chained buffers) of space to be used as an output buffer for the block.

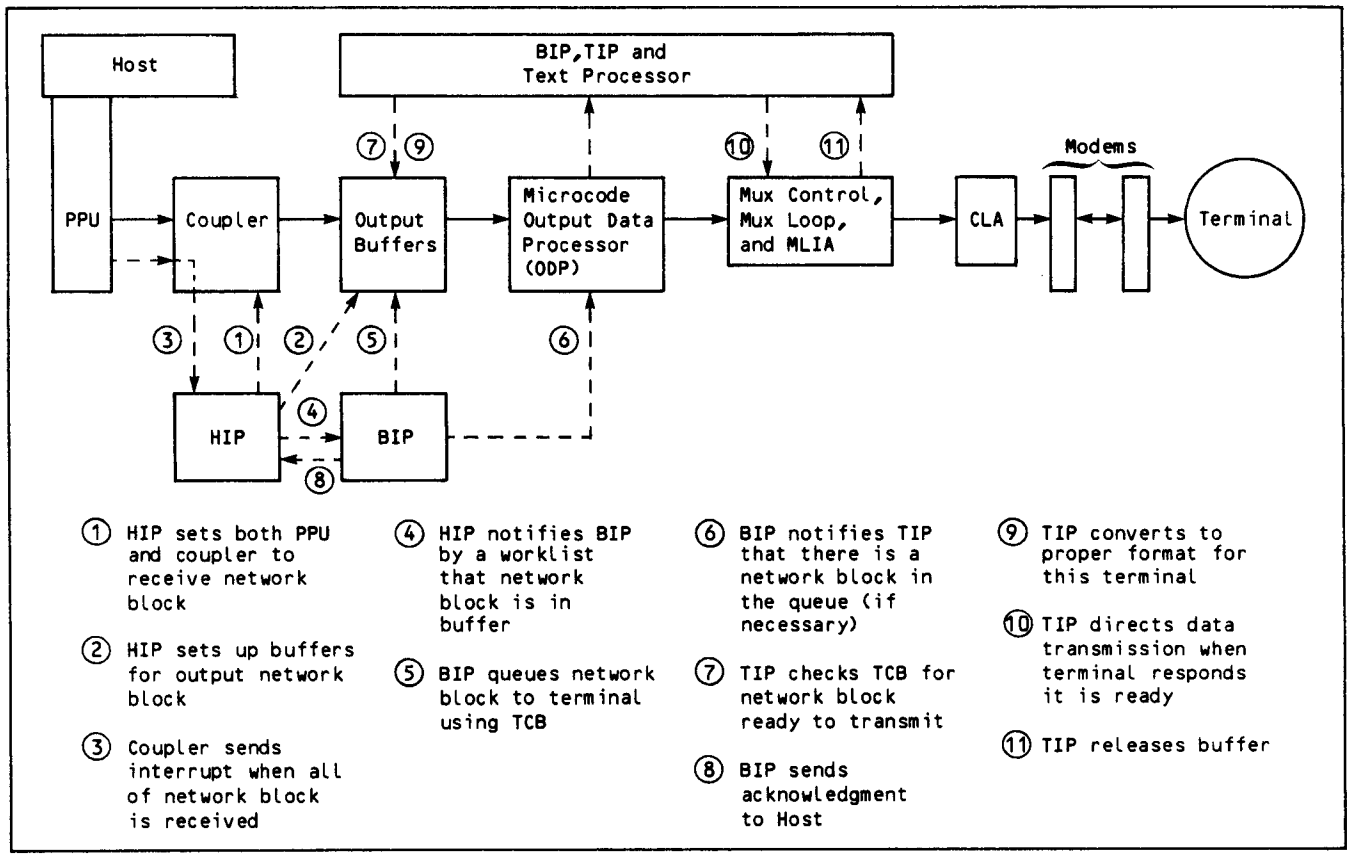


Figure 7-4. Simplified Output Processing

The network block is sent by a buffered transfer from the PPU through the coupler to the assigned buffers. The coupler causes the transmission-complete interrupt to the HIP when all of the network block has been received.

The text processor converts the network block from IVT or PRU format to the destination terminal's format (transmission blocks).

The transmission block is queued to a terminal control block where the terminal interface package (TIP) detects that data is available.

If the terminal is able to receive output data, the TIP directs the multiplex subsystem to output the message.

When a synchronous terminal has detected the end of a transmission, it sends an acknowledgment.

When the TIP knows that the data was received, the NPU terminates the output operation by sending an acknowledgment to the host and by releasing the output buffers.

Line statistics (a type of status information that includes both success and failure data, such as number of blocks and characters processed or the number of transmission errors detected) are updated for the transfer. These statistics are periodically sent upline to the host; some become accounting messages, others become hardware error log messages.

DATA PRIORITIES

Network blocks are defined to be high or low priority as a function of the device which sends or receives the data in the block. Regulation of data is needed because all network blocks require buffer space; it is possible that combined peak load demands from terminals, from the host, and from neighboring NPUs (if any) may require more buffers than the NPU can assign.

To prevent NPU stoppage for lack of buffers, the NPU is allowed to reject input on the basis of priority or because the channel seeking to input data already has its maximum share of buffers assigned. For terminals which are polled for input, regulation is handled indirectly; the NPU does not poll the terminals until more buffers are available. As current network blocks are processed and output, more buffers become available and the regulation level can change to accept data previously rejected.

Two types of regulation exist:

Regulation of upline and downline data for specific connections

Regulation of upline and downline data for entire logical links (data paths through an NPU to a host)

Connection Regulation

Data to or from a terminal device is regulated by the upline and downline network block limits assigned for the device. The NPU queues the number of input blocks specified as the upline block limit for the device. The NPU also queues the number of downline blocks queued for the device outside of the host; the portion queued by the host is established by two queuing limits, the application block limit and the downline block limit.

The application, downline, and upline block limits are initially established by the network administrator in the network definition file, using the Network Definition Language statement ABL, DBL, and UBL parameters. This process is discussed in section 2 and in the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface.

When the upline block limit for a device is reached, input regulation begins for the device. Input regulation is discussed separately for each TIP, later in this section. When the application block limit for downline data is reached, host processes perform output regulation without involving the NPU.

Levels of Logical Link Regulation

Three levels of traffic priority are defined on a logical link:

Service message traffic (priority 1): This level includes command, reply, and status information exchanged between neighboring NPUs and between NPUs and a host. Traffic at this level is suspended only after suspension of traffic at levels 2 and 3 fails to produce the needed buffer space.

High-level (priority 2): This level is intended for data associated with an interactive type of device. The size of each network block is normally small but the network block is quickly processed so that no processing delay is visible to the terminal user. Traffic at this level is suspended only after suspension of traffic at level 3 fails to produce the needed buffer space.

Low-level (priority 3): This level is intended for data associated with batch-type devices. The size of the network block is large (often a thousand bytes or more) but since no operator interaction is involved, a small delay in network block processing is acceptable. Traffic at this level is suspended first when regulation must occur to produce the needed buffer space.

The actual assignment of priorities (levels 2 and 3) to devices occurs in the network configuration file. Refer to the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface.

When the logical link used by a device becomes subject to traffic suspension at the indicated priority level, input regulation begins for the device. Input regulation is discussed separately for each TIP, later in this section.

Most downline regulation occurs through host processes, without involving the NPU. The NPU can reject downline data when buffer regulation is in effect.

TERMINAL INTERFACE PROGRAMS

There are two basic categories of Terminal Interface Programs:

Site-written TIPs

CDC-written TIPs

A site-written TIP can consist of:

Code written specifically to support a communications protocol not currently supported by a CDC-written TIP

Code added to a CDC-written TIP to support terminals that use CDC-supported protocols but have characteristics not compatible with any CDC-defined terminal class or with any CDC-defined device type

The functions and characteristics of site-written TIPs are unknown. The configurable features of communication lines, terminals, and devices supported by such TIPs are described in the Network Access Method Version 1 Network Definition Language Reference Manual described in the preface.

The released version of CCP has separate TIPs to support five communication protocols. These are:

The ASYNC TIP, supporting teletypewriter-compatible or IBM-2741-compatible devices

The X.25 TIP, supporting packet-switching network terminals using a packet assembly/disassembly (PAD) facility

The MODE4 TIP, supporting terminals and devices that use either the mode 4A or mode 4C variants of the CDC Mode 4 communications protocol

The HASP TIP, supporting terminals and devices that use the Houston Automatic Spooling Program (HASP) protocol

The BSC TIP, supporting terminals and devices that use the IBM 2780 or IBM 3780 variants of the IBM bisynchronous protocol

ASYNC TIP

The ASYNC TIP supports terminal devices connected through dedicated or dial-up asynchronous lines. Such devices normally generate IVT data to exchange in an interactive mode with a host application program. The TIP supports one terminal device per line, and expects each device to have an input and an output mechanism.

The following CDC-defined terminal classes are supported:

Terminal Class	Archetype Terminal
1	Teletype Models 33, 35, 37, 38
2	CDC 713, 751, 756
3	Reserved for CDC
4	IBM 2741
5	Teletype Model 40-2
6	Hazeltine 2000
7	CDC 752
8	Tektronix 4014, 4114

Site-defined devices in terminal classes 28 through 31 can also be supported if the site modifies the released version of the ASYNC TIP.

Each CDC-defined terminal class has a default set of terminal characteristics as shown in appendix F. These terminal characteristics can be changed from their defaults initially for each device through the network configuration file or subsequently by the terminal user and/or the application program through the use of commands.

The valid values and default parameters for commands in each of these terminal classes are shown in appendix F. When a valid value is entered from a terminal, the TIP responds with the message:

cd ACCEPTED..

where cd is the command mnemonic. When an invalid value is entered, the TIP responds with the message:

cd reason

and the command represented by cd is ignored. The field identified as reason contains a message such as DUPLICATE CHARACTER, INAPPROPRIATE VALUE, ILLEGAL TERMINAL CLASS, ILLEGAL COMMAND, or ILLEGAL VALUE, explaining the reason the command was ignored.

The ASYNC TIP supports all IVT interface commands and features except the following ones for 2741 terminals:

Break condition as user break or cancel input (the BR command)

Special editing (the SE command)

Paper tape input or output (the IN command PT and XP options, the OP command PT option)

Input and output regulation of or by devices, called device control (the IC and OC commands)

Abort output blocks (the AB command)

Input cursor positioning response (the CP command)

Block mode operation

Changing the end-of-line character code to a value other than the default delimiter (the EL or EB command X options)

Line mode cursor positioning (the EL command CP option)

Transparent mode timeout or character count delimiter, or changing the character code used as a delimiter (the DL or the XL command TO or C option, or the X and Y options)

Echoplexing (the EP command)

Protocol Assumptions

This subsection describes those aspects of TIP operation that depend upon hardware use or are implementor-determined portions of asynchronous communications protocol. These aspects include:

Line access

Line speed and stop bit use

Operating modes

Point-to-Point and Multidrop Line Access

The ASYNC TIP supports point-to-point communication line access for nonaddressable terminals. Multidrop terminal configurations are not supported.

Speeds and Stop Bits

Teletypewriter-compatible terminals (classes 1, 2, and 5 through 8) are assumed to use 2 stop bits when operating at 110 b/s and 1 stop bit when operating at speeds from 150 to 9600 b/s. 2741-compatible terminals (class 4) that operate at speeds of 134.5 b/s use 1.5 stop bits; 2741-compatible terminals that operate at speeds of 300 b/s use 1 stop bit.

Modes of Operation

The logical mode of operation of the TIP is normally half-duplex, with input given priority over output. This means that output is not started when input is active. If input occurs during output, output is stopped within a few characters, and continues with the first unsent character after input completes. The TIP attempts to deliver output whenever such data is available, unless: input is in process, a page wait condition is in force, or an automatic input mode block has been delivered to the terminal.

The only exception to this half-duplex mode of operation occurs when full-duplex operation is required by the application program (refer to the turn-on-full-duplex supervisory message described in section 3). Full-duplex operation, supporting simultaneous input and output, is functional only when transparent keyboard input is selected (refer to the XK or X options of the IN command in appendix F). When the TIP operates in full-duplex mode, device input and output control (IC and OC commands), page waiting (PG command), echoplexing (EP command), and automatic input are not effective options for the terminal user or the application program.

Supported Input and Output Mechanisms

The TIP supports a range of input and output mechanisms; only one input and one output mechanism can be active at any given time. Control of multiple output mechanisms (for example, a display and a hardcopy printer) is left up to the terminal.

When normalized mode input is in effect, the TIP supports the following input modes and mechanisms (refer to the IN command in appendix F):

Keyboard in line mode. Input is forwarded after an end-of-line or end-of-block character is recognized; available output is sent after either the end-of-line or the end-of-block character.

Keyboard in block mode (not supported for 2741 devices). Input is forwarded after an end-of-line or end-of-block character is recognized; available output is only sent after the end-of-block character. Unless the terminal is operating in full-ASCII mode (Y option of the FA command, described in appendix F), an end-of-block character immediately following an end-of-line character does not cause the TIP to send an empty line to the host.

Paper tape in block mode with a fixed end-of-block indicator sequence (not supported for 2741 devices). After last message block of queued output is delivered, the TIP sends an ASCII DC1 (X-ON) character code to start the tape reader. Input is forwarded after the end-of-line character is recognized; available output is only sent if the end-of-line character is followed by the ASCII DC3 (XOFF) character. The sequence of end-of-line and DC3 codes forms the fixed end-of-block indicator. The DC3 can be preceeded by an ASCII LF character as well; the LF is not part of the forwarded input.

When normalized mode output is to be sent, the TIP supports the following output devices (refer to the OP command in appendix F):

Display. Line folding and scrolling are assumed to be performed by the output device when page width (see PW command in appendix F) is reached.

Printer. Line folding is performed by the TIP when page width is reached. Paging is performed by the TIP when page length is reached.

Paper tape (not supported for 2741 devices). This output mechanism is treated the same as a printer, with the exception that an ASCII DC3 character is sent after the completion of every downline post-print format effector action.

In transparent mode, the keyboard or paper tape are allowed input mechanisms and the display, printer, or paper tape are allowed output mechanisms. Forwarding of input is determined by the options selected for the DL or XL commands (see appendix F). Output formatting is left up to the host application program. In transparent mode, the application program is responsible for all data interpretation, including control characters.

Terminal Code Sets and Parity

The ASYNC TIP supports teletypewriter-compatible terminal devices operating with user selectable parity set to even, odd, zero, or none. The initial parity of the device is established in the network configuration file, but can be changed by the application program and/or the terminal user; refer to the PA command in appendix F. Teletypewriter-compatible devices use the ASCII, ASCII bit-paired APL, or ASCII typewriter-paired-APL code sets.

The TIP also supports IBM 2741-compatible terminal devices with a fixed parity (set to odd). These devices use the EBCD, EBCD-APL, Correspondence Code, or Correspondence-Code-APL code sets.

The TIP converts terminal code sets to and from network ASCII codes where necessary. The terminal code set used is initially established in the network configuration file. The terminal user and/or the application program can change the code set as part of the automatic recognition process, using the AR command described in appendix F.

Normalized Mode Output

The host sends 7-bit ASCII character codes. The eighth bit (bit 7) can be anything and is ignored. If translation is needed, bit 7 is set to zero before the TIP translates the character. If translation is not needed, the bit is set to zero before going to the CLA for output. The CLA provides the parity setting appropriate for the parity option selected. This is true for all parity settings.

Transparent Mode Output

The host can send anything for bit 7 of all parity settings; the host value for bit 7 is ignored for all parity options except parity of none. Bit 7 is set to zero for parity of zero, and is set correctly for parity of odd or even. For parity of none, the entire 8 bits, as sent from the host, are sent through the CLA to the terminal.

Normalized Mode Input

The TIP sends the host 7-bit ASCII character codes with the eighth bit always set to 0. This is true for all four parity settings.

Transparent Mode Input

For parity of none, the host receives the 8 bits just as they came in on the line. For the other three parity settings, the eighth bit is set to zero.

Initial Connection

The initial line speed and code set can be explicitly set through the network configuration file or can be determined by the TIP through automatic recognition. Recognition of line speed is available in a range of 110 to 2400 b/s.

Automatic recognition of line speed is performed when a line becomes active. The TIP sets the CLA to sample the line at 800 b/s. The user has 60 seconds to enter input. The first input on the line is presumed to be a carriage return code; the bit pattern received is compared against the patterns expected for that code at supported speeds until a match is found. The user then enters more input. If the next character entered is also a carriage return, the terminal is assumed to use:

- The ASCII code and character set if the terminal is recognized as other than a 2741
- The EBCD code and character set if the terminal is using a 134.5 b/s speed, or is known to be a 2741 operating at 300 bits per second

If the second character entered is not a carriage return, it is presumed to be a right parenthesis code; the bit pattern received is compared against the patterns expected for that code in supported code sets, until a match is found.

The need for automatic recognition is established through the network configuration file. Once line speed and code set are known, the terminal connection to the NPU is complete. Subsequent connection to a host and an application program depend on the network configuration; the procedure and possible commands required are described in appendix H.

After initial connection, the automatic recognition procedure can be restarted by the terminal user without disconnecting the terminal from either the host application program or the NPU. See the AR command description in appendix F.

Disconnection

The ASYNC TIP disconnects a terminal on a dial-up line when:

- The terminal uses a line speed or code set not recognized by the TIP (after 60 seconds of attempted automatic recognition).
- The terminal user fails to complete both steps of the automatic recognition sequence within the time allowed.
- The terminal is not connected to a host and no input has been received from it for 2 minutes.
- The Communications Supervisor requests that the line or device be disabled.
- The user hangs up the phone.

The ASYNC TIP does not disconnect a terminal on a dedicated line. Input from disabled terminals or input on disabled hardwired lines is discarded. Output to disabled terminals is possible (for example, messages from the network operator). Failure to complete automatic recognition on a dedication line causes the TIP to restart the procedure.

Data Formatting in Normalized Mode

During input, the TIP detects the end of a physical line when:

A linefeed code is entered and an upline blocking factor of 0 is in use (an upline network block is generated when 0 is in use, even though the corresponding network block size is 100 characters).

The page width is greater than 56 characters and the page width is reached.

The page width is 0 or greater than 56 characters and an end-of-line or end-of-block indicator is entered.

The TIP does not discard stored characters of a physical line in response to backspace codes when the backspaces occur after the end of the physical line. The user cannot backspace across a physical line boundary to delete previously entered characters.

During output to a device with a printer mechanism (refer to the PR option of the OP command in appendix F), a physical line ends when the page width is reached or when the output contains a unit separator code, a carriage return code, or a line feed code. During output to a device with a display mechanism (DI option of the OP command), a physical line ends when the output ends or when it contains a unit separator code, a carriage return code, or a line feed code.

The TIP inserts carriage return, line feed, and idle codes when needed after the end of a physical line; refer to the CI and LI commands in appendix F.

Normalized Editing Modes

The three editing modes described in section 2 are supported for all terminal classes except terminal class 4 (2741s). For 2741s, only the complete editing and full-ASCII editing modes are supported.

In complete editing mode and special editing mode, the cancel input character is stored as data and a flag is set in the network block. The flag becomes the can bit of the application block header in the host; refer to the CN command in appendix F. After the cancel input character is entered, the TIP sends the message

DEL

to the terminal.

In special editing mode and full-ASCII mode, line feed codes and backspace characters are data and are not processed by the TIP.

In special editing mode, input of a backspace followed by a line feed code causes output of a bell code and a pointer character, followed by a backspace and a line feed. For the ASCII, EBCD, and Correspondence Code character sets, the pointer character is ! ; for the APL character sets, the pointer character is ∨.

Input Operations

Section 2 describes the general case for any interactive virtual terminal input operation. This subsection describes the special cases and subsets of those operations unique to terminals serviced through the ASYNC TIP.

When input ends, the TIP can perform cursor positioning as a response to that condition. Whether cursor positioning occurs is determined by the current setting of the CP command (refer to appendix F). The response used is determined by the current value for the CP option of the EL or EB command. For 2741 terminals, cursor positioning always occurs after input because the terminal ends each line with a newline code (NL).

The following IVT operations can be correctly supported by the ASYNC TIP only in line mode:

Cursor positioning responses at the end of each physical and logical line (refer to Y option of the CP command in appendix F); cursor positioning at the end of each block is the only option that can be supported in block mode.

Special editing mode processing of the backspace-linefeed code sequence.

For 2741 terminals, the TIP keeps track of shifts between uppercase and lowercase. Each physical line of input is presumed to start in lowercase.

For paper tape operation (refer to PT option of the IN command in appendix F), a DC3 (X-OFF) code is sent upline as data unless it follows an end-of-line indicator. For transparent paper tape operation (refer to XP option of the IN command), a DC3 (X-OFF) code following an end-of-line indicator and the end-of-line indicator itself are sent upline as data when that sequence does not delimit multiple message transparent input.

Output Operations

Two interactive virtual terminal output control operations are unique to the ASYNC TIP:

Discarding output on terminal user or application program command

Suspending output in response to device commands that perform output regulation

Discarding Output

If the terminal user enters the abort output block character (refer to the AB command in appendix F), the ASYNC TIP discards the rest of the current transmission block and the current network block. Output continues with any subsequent block received from the host.

Output Regulation

Terminals can transmit an isolated ASCII DC3 (X-OFF) code to stop output. When output control by a device is selected (refer to the Y option of the OC

command in appendix F), the TIP stops output within n characters (n depends on the system load, but is normally 2) and discards the received DC3 code.

The terminal can then transmit an isolated ASCII DC1 (X-ON) code to resume output. When the TIP receives that code, it discards the DC1 and resumes output with the next character in the current buffer.

If the terminal does not transmit an ASCII DC1 (X-ON) code, any input followed by an end-of-block or end-of-line indicator causes output to resume. The TIP discards the end-of-block or end-of-line indicator if appropropriate for the current operating mode, and the received input is sent to the host as data.

The ASYNC TIP also supports output regulation through use of the clear to send (CTS) modem signal. The TIP stops output within two characters when CTS drops, and resumes output when CTS is raised again.

Break Significance

A break condition on an asynchronous communication line exists when a null character with a framing error is detected by the CLA. Such a condition causes the TIP to stop any output in progress and wait for consecutive input. This condition has the following additional meaning to the ASYNC TIP only when normalized mode input is used (IN command KB or PT or BK options, described in appendix F).

A device can be defined such that detection of a break condition on the line is interpreted as either a user break 1 signal (refer to the B1 command in appendix F) or as a cancel input signal (refer to CN command in appendix F). This is done initially through the network configuration file and subsequently by an application program and/or a terminal user through the Y option of the BR command, described in appendix F.

If input is in process when such a break condition is detected, the condition is treated as a request to cancel input. A flag is set, and the cancelled data is sent upline to the host for discarding. The set flag becomes the can bit in the application block header. As it does when it detects the cancel character at the end of a line, the TIP sends the message

```
*DEL*
```

to the terminal.

If the terminal is idle or output is in process when such a break condition is detected, the condition is treated as a user break 1 request. A break block is created, and all data queued for output is discarded; the break block is sent upline to the host. In the host, the break block becomes a supervisory message and also sets the brk bit in the application block header of a subsequent network block.

Input Regulation

As mentioned earlier in this section under Data Priorities, the TIP must sometimes suspend input. For devices that support input control (refer to the Y option of the IC command in appendix F), input is stopped whenever a temporary suspension of input must occur. The TIP sends an ASCII DC3 (X-OFF) code to accomplish this. When regulation ends, the TIP sends an ASCII DC1 (X-ON) code to these terminals.

If input of a logical line occurs when regulation is needed, the message

```
WAIT..
```

is sent, preceeded by a DC3 (X-OFF) code. The data transmitted from the terminal is discarded. For terminals operating in block mode, a varying number of logical lines within a block might be discarded; that is, regulation can begin after the first logical line is stored and can end before the last logical line of the transmission block begins. The WAIT message is sent for each discarded line.

When data has been discarded, the message

```
REPEAT..
```

is sent as soon as regulation ends and data can again be accepted and stored by the NPU. Paper tape readers must be restarted manually.

Output from the host is sent to the terminal during input regulation, unless input was in process when regulation began.

Error Recovery

The ASYNC TIP performs no error recovery on input data; any error recovery needed must be performed by the application program in the host. If a parity error or framing error is detected during input, the TIP stores the erroneous code in the network block and sets a flag. The flag becomes the pef bit in the host application block header.

X.25 TIP WITH PAD

The X.25 TIP PAD subTIP supports asynchronous terminal devices connected through a packet-switching network (PSN) or a public data network (PDN). These networks provide a packet assembly/disassembly (PAD) access facility with separate switched virtual circuits for each device, and use a dedicated or dial-up synchronous communication line to exchange data with the NPU.

The terminal device is connected to the PDN through the PAD facility. The PAD facility connects to the terminal over a voice grade line at low or medium speeds.

X.25 devices normally generate IVT data to exchange in an interactive mode with a host application program. The TIP supports one terminal device per virtual circuit, and expects each device to have an input and an output mechanism.

The following CDC-defined terminal classes are supported:

Terminal Class	Archetype Terminal
1	Teletype Models 33, 35, 37, 38
2	CDC 713, 751, 756
3	Reserved for CDC
5	Teletype Model 40-2
6	Hazeltine 2000
7	CDC 752
8	Tektronix 4014, 4114

Site-defined devices in terminal classes 28 through 31 can also be supported if the site modifies the released version of the X.25 TIP.

Each CDC-defined terminal class connected via a PAD access facility has a default set of terminal characteristics, as shown in appendix F. Some of these characteristics differ from those the terminal class would possess if serviced through the ASYNC TIP. These terminal characteristics can be changed initially for each communication line through the network configuration file or subsequently by the terminal user and/or the application program through the use of commands.

The valid values and default parameters for commands in each of these terminal classes are shown in appendix F. When a valid value is entered from a terminal, the TIP responds with the message:

cd ACCEPTED..

where cd is the command mnemonic.

When an invalid value is entered, the TIP responds with the message:

cd reason

and the command represented by cd is ignored. The field identified as reason contains a message such as DUPLICATE CHARACTER, INAPPROPRIATE VALUE, ILLEGAL TERMINAL CLASS, ILLEGAL COMMAND, or ILLEGAL VALUE, explaining the reason the command was ignored.

The X.25 PAD subTIP supports all IVT interface commands and features except the following:

Special editing (the SE command)

Paper tape input or output (the IN command PT and XP options, the OP command PT option)

Keyboard input options KB and XK (the IN command) are not recognized because only the X option is needed

Input and output regulation of or by devices, called device control (the IC and OC commands)

Abort output blocks (the AB command)

Cancel input using a break condition key (the BR command)

Line mode cursor positioning (the EL command CP option)

Changing the end-of-block indicator (the EB command X option)

Transparent mode timeout delimiter (the DL or XL command TO option)

Echoplexing (the EP command)

Protocol Assumptions

This subsection describes those aspects of TIP operation that depend upon hardware use or are implementor-interpreted portions of X.25 communications protocol. These aspects include:

Line access

Frame and packet characteristics

Operating mode

Point-to-Point and Multidrop Line Access

The X.25 TIP supports point-to-point communication lines but allows multiple virtual circuits to share the line. Multidrop terminal configurations within a virtual circuit are not supported.

Frame and Packet Characteristics

A public data network (PDN) uses the CCITT X.25 protocol to transfer data over a high-speed communication line called a link. A PDN can have one or more links to a single NPU.

Each link from a PDN can carry one or more virtual circuits. Virtual circuits can be either switched or permanent; terminals using a PAD always use switched virtual circuits (refer to the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface). Each terminal device using the link uses a separate virtual circuit on the same physical communications line; the X.25 TIP multiplexes the virtual circuits as a software function.

The PAD access protocol is defined by the following CCITT recommendations:

Recommendation X.3 defines the conversion of asynchronous character-by-character ASCII data to a synchronous packet format. This recommendation provides options which govern PAD operation; these options are selected when the user subscribes to the PDN.

Recommendation X.28 defines these options for the terminal.

Recommendation X.29 defines these options for the NPU.

Because some of these selections affect network operation, CDC supports a set of standard PAD reference selections common to the X.25 networks listed in the Network Access Method Version 1 Network Definition Language Reference Manual (see preface). These selections are not required; however, the options chosen must not conflict with the commands supported for the interactive virtual terminal interface.

Upline, the PAD facility consolidates terminal data into packets, adding prefix and suffix information when necessary. Packets are placed in frames by the PDN and transmitted over the link to the NPU.

The TIP discards all frame and packet prefix and suffix information, and forwards the terminal data to the host as network blocks of the appropriate block type, depending on the upline block size and the presence of end-of-line or end-of-block indicators. The end-of-packet sequence bit (an M bit value of 0) is interpreted as an end-of-block indicator.

Downline, the TIP collects IVT format data into packets. Each network block becomes one or more packets; the continuation bit (the M bit) of the packet is set to one in all packets except the last created from a single network block; messages continued across network blocks are not marked as continuation packets. The TIP then assembles the packets into frames for transmission over the link. Each frame contains data for only one virtual circuit. The PAD access facility disassembles the packets into characters of data for the terminal.

Mode of Operation

The PAD subTIP is insensitive to line speeds. The logical mode of operation of the PAD subTIP is half-duplex. The subTIP delivers output whenever such data is available, unless: input is in progress, a page wait condition is in force, or an automatic input mode block has been delivered to the terminal.

There is no exception to this half-duplex mode of operation. When full-duplex operation is requested by the application program, the PAD subTIP cannot honor the request (refer to the turn-on-full-duplex supervisory message described in section 3).

Supported Input and Output Mechanisms

The PAD subTIP supports a range of input and output mechanisms; only one input and one output mechanism can be active at any given time. Control of multiple output mechanisms (for example, a display and a hardcopy printer) is left up to the terminal.

When normalized mode input is in effect, the PAD subTIP supports only one input mode and mechanism (refer to the IN command in appendix F), the keyboard in block mode. Input is forwarded after an end-of-line or end-of-block indicator is recognized (the end-of-packet continuation bit, bit M, is always an end-of-block indicator). If an end-of-block indicator in addition to the M bit is defined, that character is discarded. Available output is only sent after an end-of-block indicator is received.

When normalized mode output is to be sent, the PAD subTIP supports the following output devices (refer to the OP command in appendix F):

Display. Line folding and scrolling is assumed to be performed by the output device when page width (see PW command in appendix F) is reached.

Printer. Line folding is performed by the TIP when page width is reached. Paging is performed by the TIP when page length is reached (se PL command in appendix F).

In transparent mode, the keyboard is the allowed input mechanism and the display or printer are allowed output mechanisms. Forwarding of input is determined by the options selected for the DL or XL commands (see appendix F). Output formatting is left up to the host application program. In transparent mode, the application program is responsible for all data interpretation, including control characters.

Terminal Code Sets and Parity

The PAD subTIP supports teletypewriter-compatible terminal devices operating with user selectable parity set to even, odd, zero, or none. The initial parity of the device is established in the network definition file, but can be changed by the application program and/or the terminal user; refer to the PA command in appendix F. Devices accessing the NPU through PAD acess facilities use the ASCII code set.

Normalized Mode Output

The host sends 8-bit ASCII character codes. The eighth bit (bit 7) can be anything and is ignored. Bit 7 of the host output is replaced before going to the CLA for output; the TIP supplies a value for bit 7 that is appropriate for the selected parity option. This is true for all parity settings.

Transparent Mode Output

The host can send anything for bit 7 of all parity settings; the host value supplied for bit 7 is ignored for all parity options except a parity of none. The TIP sets bit 7 to zero for parity of zero, and correctly for parity of odd or even. For parity of none, the entire 8 bits, as sent from the host, are sent through the CLA to the terminal.

Normalized Mode Input

The TIP sends the host 8-bit ASCII character codes with the eighth bit always set to 0. This is true for all four parity settings.

Transparent Mode Input

For parity of none, the host receives the 8 bits just as they came in on the line. For the other three parity settings, the eighth bit is set to zero. Parity is not checked for correctness; parity correctness is determined by the PAD access facility before the data is received by the NPU.

Initial Connection

Automatic recognition through the TIP is not supported. Code and character set are fixed aspects of the PAD facility used by the terminal device.

The X.25 TIP supports either data terminal equipment (DTE) or data channel equipment (DCE) termination protocol. The X.25 TIP can also accept service charges for incoming calls. Both aspects of virtual circuit initialization must be known before terminal connection to the NPU can be completed.

Protocol terminating location information and service charge acceptance requests are supplied by the PDN and must agree with information supplied by the network administrator in the network configuration file. Protcol termination location, charge acceptance, and appropriate response information is described in the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface.

When terminal connection to the NPU is complete, subsequent connection to a host and an application program occur. That connection process depends on the network configuration; the procedure and possible commands required are described in appendix H.

Disconnection

The X.25 TIP clears a switched virtual circuit when the Communications Supervisor disables the terminal. The X.25 TIP does not clear a virtual circuit or disconnect a dedicated communication line. Output to disabled terminals (for example, a message from the network operator) is not possible.

Data Formatting in Normalized Mode

During input, the TIP detects the end of a physical line when:

A linefeed code is entered and an upline blocking factor of 0 is in use (an upline network block is generated when 0 is in use, even though the corresponding network block size is 100 characters)

The page width is 0 or greater than 56 characters and an end-of-line or end-of-block indicator is entered

The TIP does not discard stored characters of a physical line in response to backspace codes when the backspaces occur after the end of the physical line. The user cannot backspace across a physical line boundary to delete previously entered characters.

During output to a device with a printer mechanism (refer to the PR option of the OP command in appendix F), a physical line ends when the page width is reached or when the output contains a unit separator code, a carriage returncode, or line feed code. During output to a device with a display mechanism (DI option of the OP command), a physical line ends when the output ends or when it contains a unit separator code, a carriage return code, or a line feed code.

The TIP inserts carriage return, line feed, and idle codes when necessary after the end of a physical line; refer to the CI and LI commands in appendix F.

Normalized Editing Modes

Only two of the editing modes described in section 2 are supported: complete editing and full-ASCII editing.

In complete editing mode, the cancel input character is stored as data and a flag is set in the network block. The flag becomes the can bit of the application block header in the host; refer to the CN command in appendix F. The TIP sends the message

```
*DEL*
```

to the terminal.

In full-ASCII mode, physical lines are not recognized as input boundaries for backspacing.

Output Operations

Some PAD access facilities recognize an ASCII DC3 (X-OFF) code as a control signal to interrupt output. A subsequent ASCII DC1 (X-ON) code serves to resume output from the PAD. This usage is not part of the X.25 protocol. If the PAD does not support output control, any DC3 (X-OFF) or DC1 (X-ON) code sent by the terminal is received by the TIP, treated as data, and sent upline to the host application program.

Break Significance

A break condition on an X.25 communication line is a selectable option of the PAD facility. A device can be defined such that occurrence of a break condition on the line is interpreted as a user break 1 signal (refer to the B1 command in appendix F). This is done initially through the network configuration file and subsequently by an application program and/or a terminal user through the Y option of the BR command, described in appendix F.

The TIP discards all data queued for output to the PDN. The TIP creates a break block and sends it upline to the host. In the host, the break block becomes a supervisory message and also sets the brk bit in the application block header of a subsequent network block.

The TIP retains subsequently received downline data until input from the terminal occurs. The PAD sub-TIP notifies the PAD so that subsequent input is allowed.

The PAD discards all downline data queued from the TIP but not received by the terminal. The PAD might not discard upline data queued from the terminal within the PDN.

Because the X.25 protocol does not allow the TIP to determine if input was in process when the break condition occurred, the TIP cannot interpret a break condition as a request to cancel input.

Input Regulation

As mentioned earlier in this section under Data Priorities, the TIP must sometimes suspend input. When regulation is needed, the TIP stops acknowledging incoming packets. Unacknowledged packets are retained and are acknowledged once regulation ends. At some point while regulation is occurring, the PDN will stop sending packets because the packet window (the permitted number of unacknowledged packets) is reached. The packet window is described in the Network Access Method Version 1 Network Definition Language Reference Manual mentioned in the preface.

The terminal user is not informed by the TIP when regulation is in process. Packets that cannot be sent to the TIP from the PDN are presumed to be queued by the PDN so that no data is lost.

Output from the host is sent to the terminal during input regulation, unless input was in process when regulation began.

Some PDN PAD access facilities support DC3 (X-OFF) and DC1 (X-ON) codes as device input control codes. This feature is not part of the X.25 protocol and is not recognized or controllable through the PAD subTIP.

Error Recovery

Because the TIP does not check parity on input, parity errors are not detected or reported to the host for corrective action.

At the packet level, the TIP checks for missing input packets. A missing packet causes a channel reset. Other packets travelling upline for the virtual circuit are discarded; no action is taken for packets going downline.

The action taken by the PDN in response to a channel reset depends on the specific PDN. If the PDN does not retransmit the packet, the data is lost. The TIP does not inform the terminal user of a channel reset.

If the TIP receives a reset packet from the PDN, it reinitializes the channel in both directions. All packets travelling upline and downline on the same circuit are discarded. The TIP does not retransmit packets in response to a reset packet and does not inform the terminal user that upline or downline data was discarded. In the current release, the TIP does not notify the host application program of a channel reset for either cause.

At the link level, the TIP checks for bad frame content, for missing frames, and for bad frame formats. If such an error is recoverable (the frame is identified and was retained for retransmission), the frame is resent. Otherwise, the link is reset. The action taken for a link reset is dependent upon the specific PDN. The TIP does not retransmit downline packets after a link reset and does not notify either the host application program or the terminal user.

MODE4 TIP

The MODE4 TIP supports terminals using CDC Mode 4 protocol connected through dedicated or dial-up synchronous lines. Such terminals contain devices that generate IVT data to exchange with a host application program in an interactive mode, and devices that exchange PRU data with a host application program in a batch mode.

The TIP expects each terminal to have at least one interactive device. The interactive device must have an input and an output mechanism.

The terms cluster, terminal, and device have varying definitions within mode 4 documentation. This subsection uses these definitions:

> A terminal is a group of devices with a common cluster address.
>
> A cluster is a group of devices with a single cluster controller.
>
> A device is a separately accessible unit of input or output equipment, such as a console, a card reader, or a line printer. Within the network, a device is separately addressable, regardless of whether it has a separate hardware address within the terminal. For mode 4C terminals, devices have separate terminal addresses; for mode 4A terminals, devices use a fixed terminal address (that of the console device) and an interactive or batch mode of access.

Cluster addresses and terminal addresses are described in the Network Access Method Version 1 Network Definition Language Reference Manual, listed in the preface. The maximum number of terminals on a communication line and the maximum number of devices within each terminal also is described in that manual.

A CDC-defined mode 4A terminal has one console, one card reader, and one line printer. A CDC-defined mode 4C terminal can have one or more consoles and line printers.

The following CDC-defined terminal classes are supported:

Terminal Class	Archetype Terminal
10	CDC 200 User Terminal (mode 4A)
11	CDC 714-10/20 display terminal (mode 4C)
12	CDC 711-10 display terminal or Tektronix 4014 terminal (mode 4C)
13	CDC 714-30 display terminal (mode 4C)
15	CDC 734 and 731 remote batch terminal (mode 4A)

Site-defined devices in terminal classes 28 through 31 can also be supported if the site modifies the released version of the MODE4 TIP.

Each CDC-defined terminal class has a default set of terminal characteristics as shown in appendix F. These terminal characteristics can be changed from their defaults initially for each terminal or device through the network configuration file or subsequently by the terminal user and/or the application program through the use of commands.

The valid values and default parameters for commands in each of these terminal classes are shown in appendix F. When a valid value is entered from the console, the TIP responds with the message:

cd ACCEPTED..

where cd is the command mnemonic. When an invalid value is entered, the TIP responds with the message:

cd reason

The command represented by cd is ignored. The field identified as reason contains a message such as DUPLICATE CHARACTER, INAPPROPRIATE VALUE, ILLEGAL TERMINAL CLASS, ILLEGAL COMMAND, or ILLEGAL VALUE, explaining the reason the command was ignored.

The MODE4 TIP supports all of the IVT interface commands and features except the following:

Automatic recognition restart (the AR command)

Parity selection (the PA command)

Special editing (the SE command)

Paper tape input or output (the IN command PT and XP options, and the OP command PT option)

Console printing mechanism selection (the OP command PR option)

Input and output regulation of or by devices, called device control (the IC and OC commands)

Abort output blocks (the AB command)

Cursor positioning (the EL and EB command CP option or the CP command)

Line mode operation (the EL command X option)

Changing the end-of-block character code to values other than the default (the EL and EB command Y option)

Transparent mode timeout or character count delimiter, or changing the character code used as a delimiter (the DL or the XL command TO or C option, or the X and Y options)

Echoplexing (the EP command)

Break condition as user break or cancel input (the BR command)

Idle character insertion at end of physical lines (the CI and LI commands)

Backspace processing (the BS command)

Protocol Assumptions

This subsection describes those aspects of TIP operation that depend upon hardware use or are implementor-interpreted portions of CDC Mode 4 communications protocol. These aspects include:

Unsupported protocol features

Polling

Toggle bit use

Line access

Operating mode

Unsupported Protocol Features

The MODE4 TIP does not use or support the following Mode 4 protocol message types:

Diagnostic write

Alert

Initialize command or request

Terminal disconnect

Network disconnect

Polling

The MODE4 TIP polls each mode 4A console and requests status from mode 4C clusters on a timed basis. The TIP uses a fast and slow method of service based on whether there was any recent data traffic from or to the terminal. Mode 4C consoles are polled only when the cluster controller indicates an active read request in the cluster's status response.

Toggle Bit Use

The toggle bit (bit 4 of the terminal address) is used during recovery of lost write messages or lost responses to write messages. The toggle bit value returned from a mode 4 terminal differs according to the terminal class.

Terminals in classes 10, 11, 13, and 15 return the toggle state of the last correctly received write message. These terminals do not distinguish the type of message to which the terminal is responding when they set or clear the toggle bit.

Terminals in class 12 return the toggle state of the last received message. These terminals return the toggle state used in poll messages, status requests, and so forth.

To compensate for this difference, the TIP initializes the toggle state of the terminal by sending a clear write message. This guarantees delivery of the first output transmission block.

When the TIP polls for a lost terminal response to a write message, the TIP sets the toggle state opposite to that of the last correctly received

write message. If the toggle bit in the response is the same as in the polling message, the write message is sent again. This procedure guarantees that all write messages are correctly received by the terminal. No blocks are duplicated for terminals except 711s, because:

Blocks are only sent once to terminals in classes 10, 12, 13 and 15.

The Tektronix 4014 discards a write message with a toggle state that is the same as the previous write message.

Duplication of output during correction of a lost write message condition cannot be avoided at 711 devices. The 711 device does not maintain the correct toggle state and does not discard duplicate blocks.

Point-to-Point and Multidrop Line Access

Mixtures of mode 4A and mode 4C terminals are supported on multidrop lines. Mixtures of ASCII and External BCD terminals on the same line are also supported. Refer to the Network Access Method Version 1 Network Definition Language Reference Manual for information on the support possible for automatic recognition and fixed configuration communication lines.

Operating Mode

The MODE4 TIP is insensitive to line speeds.

The logical mode of operation of the TIP is half-duplex for each device, with console input given priority over batch output. The TIP attempts to deliver console output whenever such data is available, unless: a page wait condition is in force, or an automatic input mode block has been delivered to the terminal.

The end of an input transmission block is signified by an ETX character code. This end-of-block indicator is part of the Mode 4 protocol and cannot be changed.

Supported Input and Output Mechanisms

The TIP supports a range of input and output devices and mechanisms. Console devices have a keyboard as an input mechanism and a display as an output mechanism.

CDC-defined mode 4A batch devices are either card readers or impact line printers. Such devices are serviced sequentially with the console.

CDC-defined mode 4C batch devices are either impact printers or nonimpact printers. Such devices are serviced concurrently with the console.

Sequential Operation

For mode 4A terminals, only one interactive or batch input or output mechanism can be active at any given time. Input and output are therefore sequential operations.

The TIP suspends console operations when either the card reader becomes active or data is available for the line printer. Batch input and output blocks alternate (are interleaved) until one of the following events occurs:

Output arrives for the console from the host.

The card reader or printer is stopped (or suspended).

The card reader stops and no printer data is available for 5 seconds.

Console input is attempted (the batch interrupt key is used).

In the first three cases, the TIP issues a clear write to clear the terminal buffer. In the last case, the TIP polls the console for input data.

Concurrent Operation

For mode 4C terminals, all devices can be active at a given time, but only the input or output mechanism of the console device can be active at a given time. Input and output are therefore concurrent operations for batch devices but sequential operations for the console. Interactive operations can be concurrent with batch operations.

Card Reader Operation

Card reader operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Start input

Resume input

Abort input

The TIP sends command blocks to the host to:

Report a not ready status returned by the card reader

Report an interruption in batch operation

Report incomplete card images (fewer than 80 characters)

Report incomplete input files (when the last card read is not the end of information)

Report the end of the input stream

Report the number of cards read

Report the end of file

A mode 4 card reader becomes ready for input when the host sends the TIP a start input command block. The TIP then polls the terminal for input.

Card input begins with the first card read. A card input file begins when the beginning-of-information is detected. The TIP expects a nonblank data card as the beginning-of-information initially or following an end of information card. The TIP discards blank cards, end-of-record cards, or end-of-information cards read when it is expecting the beginning of information.

When the beginning of information is located, the TIP converts subsequent data into a NOS operating system job file in 6-bit display code. Conversion from 026 or 029 punch patterns is performed in the terminal hardware, so transparent card input (binary cards) or changes in punch translation cannot be supported.

The TIP converts the received card data from either External BCD code or ASCII code to 6-bit display code. Trailing blanks are discarded unless the last nonblank character is a colon; one blank is preserved or inserted when the last character is a colon or the card contains an odd number of significant characters. For hosts using a 64-character set, this convention produces an 82-character record when the character in column 80 is a colon.

Every card image becomes a zero-byte terminated record. These records become a network block equivalent to one, two, or three physical record units (PRUs); a block equivalent to one PRU contains 640 6-bit characters or equivalent zero-byte terminator bytes, left-justified in an 8-bit character byte. A record is continued in a subsequent network block when necessary. In the host, PIP removes the two low-order bits and packs the characters into actual PRUs.

The TIP recognizes /*EOR and 7/8/9 multipunch cards as end-of-record indicators for NOS system logical records. It converts any numbers found in columns 2 and 3 or 6 and 7 to octal record level numbers. When necessary, short or zero-length PRUs are created.

The TIP recognizes /*EOI and 6/7/8/9 multipunch cards as end-of-information indicators for NOS files.

Printer Operation

Line printer operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report an interruption in batch operation (mode 4A only)

Report a printer not ready condition (mode 4A only)

Report when a file limit is exceeded

Report when a printer message (PM) is received

Report the number of lines printed

Report the end of file

The TIP begins output to a printer when it has output queued for the device. Data received from the host is packed as line images in PRU-sized network blocks (the host PIP routine strips zero byte terminators and replaces them with hexadecimal FF code bytes). The TIP discards trailing blanks on a print line.

6-bit display code is converted to either External BCD or ASCII as appropriate for the terminal. ASCII print files are converted from 7-bit codes right-justified in 8-bit bytes to the codes supported for normalized mode data (see Terminal Code Sets and Parity). This conversion includes stripping the unneeded bit 7 (the host PIP routine strips the unneeded bits 8 through 11 prior to downline transmission). The parity bit is appropriately set during output.

Console input or output interrupts printing at a mode 4A terminal; the TIP reports this condition to the application program through a command block. Mode 4C printer busy and not-ready conditions are not reported to the host and printing continues when the condition clears.

Once started, the TIP continues output to mode 4C printers unless stopped or interrupted by a command block from the host or a printer message (PM carriage control sequence in the print file). Any mode 4C or mode 4A printer that is interrupted, stopped, or not ready can be restarted by a command block from the host.

The TIP supports both mode 4C impact and nonimpact printers. The impact printer supports page eject and can be controlled through carriage control character processing. The nonimpact printer does not support page eject, so the TIP simulates that effect when it processes carriage control characters. This simulation uses the defined page length of the device (from the network definition file) to calculate the number of lines remaining to the bottom of the page; the TIP inserts the calculated number of linefeeds into the output, then adds six more to space over the simulated page boundary.

Print Message Processing

When the print file contains a line beginning with the characters PM, the TIP stops the printer, notifies the application program through a command block, and sends the data from the print file line to the application program.

Normalized Mode Console Operations

When normalized mode input is in effect from a console, the TIP supports only one input mode and mechanism (refer to the IN command in appendix F): the keyboard in block mode. Input is forwarded after an end-of-line or end-of-block character is

recognized; available output is only sent after the end-of-block character. When input ends, the TIP positions the cursor to the lefthand margin of the next line on the display and unlocks the keyboard.

When normalized mode output is sent to a console, the TIP supports only one output mechanism, the display. Line folding and scrolling is assumed to be performed by the output mechanism when page width (see PW command in appendix F) is reached. When output ends, the TIP ensures that the cursor is positioned at the lefthand margin of the next line on the display. Page waiting is supported.

Transparent Mode

Transparent mode input and output are only supported for the display console. The TIP adds the line protocol frame control characters to output and removes them from input.

The TIP does not poll a terminal for console input after transparent mode input is received, until subsequent output is delivered to the console.

In transparent mode, the application program is responsible for handling all data interpretation, including any embedded IVT command or feature control characters. The application program must properly position the cursor at the end of input or output, and must unlock the keyboard when necessary.

The TIP does not perform clear write or reset write operations at the beginning of transparent mode output. The device's transmission buffer is therefore not cleared and reset unless the proper control codes are part of the transparent output.

Terminal Code Sets and Parity

The MODE4 TIP supports terminals operating with a fixed parity, set to odd. Mode 4A terminals use the ASCII or External BCD code sets; mode 4C terminals use the ASCII code set.

The TIP converts terminal code sets to and from network ASCII codes where necessary. No ASCII control codes exist in normalized output to External BCD terminals; the TIP replaces all nongraphic characters with blanks and folds all lowercase characters to equivalent uppercase characters (see appendix A). The following ASCII codes effectively do not exist in normalized output to ASCII terminals:

Control codes below the hexadecimal value 8 (these are converted to blanks)

The control codes US and CR (hexadecimal 1F and 0D; these are converted to newline sequences)

The control code LF (hexadecimal 1A; the TIP removes this code from the output)

In transparent mode, the TIP does not convert or delete characters in data sent to either External BCD or ASCII terminals.

The terminal code set used is initially established in the network configuration file. If the terminal supports the feature and alternate configurations exist in the network configuration file, the terminal user can change the code set prior to the automatic recognition process and initial connection to the network.

Normalized Mode Output

When the host sends ASCII character codes, the eighth upper bit (bit 7) can be anything and is ignored. If translation is needed, the value supplied by the host as bit 7 is ignored when the TIP translates the character. If translation is not needed, the bit is reset by the CLA as necessary. The CLA sets the bit to give the character odd parity.

Transparent Mode Output

The host can send anything for bit 7. Bit 7 is set correctly by the CLA for odd parity to ASCII or External BCD code terminals. The entire 8-bit byte sent from the host is never sent through the CLA to the terminal.

Normalized Mode Input

When the TIP sends the host ASCII character codes, the eighth bit is always set to 0.

Transparent Mode Input

The eighth bit is set to zero. The remaining 7 bits are sent to the host just as they came in on the communication line.

Initial Connection

The initial terminal class, protocol variant (mode 4A or mode 4C), cluster address, terminal address, device type, and code set can be explicitly set through the network configuration file or can be determined by the TIP through automatic recognition. The need for automatic recognition is established through the network configuration file.

Automatic recognition is performed when a line becomes active. The user must first press the transmission key (SEND or ETX). This action stores a read message in the cluster address for the terminal.

When the line becomes active, the TIP polls to find an active terminal. All cluster addresses are polled. If the first poll does not locate an active cluster address, a second poll occurs before the cluster is classified as failed hardware. Each poll is timed out for 1 to 1.5 seconds.

When an active terminal is found, the TIP polls that cluster address for a read message; the read message contains an escape code that indicates the code set used by the terminal. If the terminal uses External BCD, the automatic recognition process is complete; if the terminal uses ASCII, the terminal configuration is determined.

The TIP sends a configuration request to determine the devices present. If it receives error responses or no response after two attempts, the terminal is mode 4A. If the TIP receives a read response, the terminal is mode 4C and the terminal addresses and device types of the individual devices are then determined.

The information received is passed to the Communications Supervisor in the host. The Communications Supervisor compares it against the information known for all possible terminals using the line, until a match is found. The Communications Supervisor then reports the configuration to the TIP. The mode 4C console device controlling a printer is the console with the same cluster address in the network configuration file that was the last console defined before the printer was defined.

Once the addressing information and the code set are known, the terminal connection to the NPU is complete. Subsequent connection of console and batch devices to a host and an application program depend on the network configuration; the procedure and possible commands required are described in appendix H. Batch devices are always connected to the host to which the owning console is connected.

Disconnection

The MODE4 TIP disconnects a terminal on a dial-up line when:

The terminal uses automatic recognition information not recognized by the TIP.

The terminal is not connected to a host and no input has been received from it for 2 minutes.

The Communications Supervisor requests that the line or terminal be disabled.

When a console is disconnected from a host, the TIP disconnects any batch device for which that console is the owning device.

The MODE4 TIP does not disconnect a terminal on a dedicated line. Input from disabled terminals or input on disabled hardwired lines is not requested. Output to disabled terminals is possible (for example, messages from the network operator).

Data Formatting in Normalized Mode

During input, the TIP detects the end of a physical line when:

An end-of-line or end-of-block indicator is entered

During output to a console device with a display mechanism, a physical line ends when the output ends or when it contains a carriage return or unit separator code.

Normalized Editing Modes

Only two of the editing modes described in section 2 are supported: complete editing and full-ASCII editing.

In complete editing mode, the cancel input character is stored as data and a flag is set in the network block. The flag becomes the can bit of the application block header in the host; refer to the CN command in appendix F. The TIP sends the message

DEL

to the terminal after the cancel input character is received.

In full-ASCII mode, SOH, ETX, and end-of-line sequences are sent to the host as data. Full-ASCII mode is similar to transparent mode, except the TIP continues its normal polling and protocol processing procedures.

Input Regulation

As mentioned earlier in this section under Data Priorities, the TIP must sometimes suspend input. For batch devices, polling ceases and input is stopped between transmission blocks whenever a temporary suspension of input must occur. For console devices, polling ceases.

Output from the host is delivered to the terminal during input regulation. Polling and stopped input resumes as soon as regulation ends and data can again be stored by the NPU.

Error Recovery

The MODE4 TIP performs error recovery on input data by requesting retransmission of erroneous blocks from the terminal. The TIP detects the following errors:

An error indication from the terminal

No response from the terminal

An invalid address from the terminal

An invalid protocol response from the terminal

A character parity error

Data Carrier Detect signal dropped during input

An invalid MTI

An invalid or missing E code

A missing ETX or an ETX in an illegal sequence

A longitudinal parity check error

The TIP determines whether the error occurred in the cluster controller or the device, and attempts to correct the error by restarting the input/output sequence in which it occurred. This restart is called short term recovery. Short term recovery is attempted a fixed number of times (the default is 10); then the cluster or device is treated as failed hardware and long term recovery procedures begin.

If a parity error is detected during input, the TIP requests retransmission. The flag that becomes the pef bit in the host application block header is not used; data containing parity errors is discarded and replaced by correct data resent from the terminal.

Long term recovery differs for mode 4A and mode 4C terminals, and for cluster controller or device failures. For mode 4A terminals, cluster controller and device failure are identical, so separate device recovery is unnecessary; for mode 4C terminals, cluster controller and device failure are separate events.

Times indicated below are approximate. Only one attempt occurs during each cycle of servicing all the clusters and devices on the line, so that recovery attempts do not interfere with response time for other terminals using the line.

Cluster recovery on communication lines is attempted every few seconds (default is 10 seconds for point-to-point communication lines and 50 seconds for multidrop lines). For mode 4A terminals, the TIP sends a clear write message to the console; for mode 4C terminals, the TIP sends a status request message to the cluster.

If the TIP gets a valid response, the cluster controller is operational. For mode 4C terminals, the TIP then performs recovery attempts on all devices within the cluster.

Device recovery attempts occur every few seconds (default is 50 seconds). Device recovery is not attempted if the entire cluster must undergo recovery.

Device recovery consists of the TIP sending a clear write message to each mode 4C console or an empty write message to each mode 4C printer. If the TIP receives a valid response, the device is operational.

HASP TIP

The HASP TIP supports bisynchronous terminals using HASP multileaving protocol connected through dedicated or dial-up synchronous lines. Such terminals contain devices that generate IVT data to exchange with a host application program in an interactive mode, and devices that exchange PRU data with a host application program in a batch mode.

The TIP supports one terminal per line, but can support up to 22 devices per terminal. The TIP expects each terminal to have one interactive device, a console. The console must have an input and an output mechanism.

The terms terminal and device have varying definitions within HASP documentation. This subsection uses these definitions:

A terminal is equivalent to a workstation and is a set of devices with a single point of access to a communication line.

A device is a separately accessible unit of input or output equipment, such as a console, a card reader, a card punch, a plotter, or a line printer. Within the network, a device is separately addressable. For HASP terminals, all devices except the console have separate stream numbers. Each device is addressed by device type and stream number.

Stream numbers are described in the Network Access Method Version 1 Network Definition Language Reference Manual, listed in the preface.

A CDC-defined HASP terminal has one console. It can also have up to seven card readers, up to seven line printers, and up to seven card punches or plotters.

The following CDC-defined terminal classes are supported:

Terminal Class	Archetype Terminal
9	HASP terminal supporting preprint format control
14	HASP terminal supporting only postprint format control

Site-defined devices in terminal classes 28 through 31 can also be supported if the site modifies the released version of the HASP TIP.

Each CDC-defined terminal class has a default set of terminal characteristics as shown in appendix F. These terminal characteristics can be changed from their defaults initially for each communication line through the network configuration file or subsequently by the terminal user and/or the application program through the use of commands.

The valid values and default parameters for commands in each of these terminal classes are shown in appendix F. When a valid value is entered from the console, the TIP responds with the message:

```
cd ACCEPTED..
```

where cd is the command mnemonic. When an invalid value is entered, the TIP responds with the message:

```
cd reason
```

The command represented by cd is ignored. The field identified as reason contains a message such as DUPLICATE CHARACTER, INAPPROPRIATE VALUE, ILLEGAL TERMINAL CLASS, ILLEGAL COMMAND, or ILLEGAL VALUE, explaining the reason the command was ignored.

The HASP TIP supports only the following IVT interface commands:

User break 1 and user break 2 (see the B1 and B2 commands in appendix F)

Cancel input (see the CN command in appendix F)

Change network control character (see the CT command in appendix F)

Display terminal characteristics (see the CH command in appendix F)

Lockout unsolicited console messages (see the LK command in appendix F)

Change console page width (see the PW command in appendix F)

Sending a message to the network operator (see the MS command in appendix F)

Host node selection (see the HN command in appendix G)

Host availability display (see the HD command in appendix G)

Protocol Assumptions

This subsection describes those aspects of TIP operation that depend upon hardware use or are implementor-interpreted portions of HASP communications protocol. These aspects include:

Stream identification

Filler use to maintain communication when data is not available for transmission

Terminal transparent transmissions

Line access

Operating mode

Stream Identification

The HASP TIP assumes that a network-assigned stream number between 1 and 7 corresponds to bits in the function control sequence (FCS). If the bits in the FCS are numbered from left to right, the following correspondences exist:

Card reader streams 1 through 4 are bits 4 through 7; card reader streams 5 through 7 are bits C through E.

Line printer streams 1 through 4 are bits 4 through 7; line printer streams 5 through 7 are bits C through E.

Card punch or plotter streams 7 through 5 are bits 5 through 7; card punch or plotter streams 4 through 1 are bits C through F.

The stream number algorithm allows separate flow control when more than eight printers, punches, and plotters are configured.

Filler Use

The TIP uses an ACK0 acknowledgment block as an idle block filler to transmit when no data is available to send downline. This idle block is transmitted at least every 2 seconds to prevent timeout on the communication line.

Terminal Transparent Transmissions

Two forms of terminal input and output are supported:

Terminal nontransparent transmissions

Terminal transparent transmissions

In either form, the TIP recognizes two modes of data:

Nontransparent network data

Transparent network data

The HASP TIP accepts terminal transparent transmissions beginning with the sequence DLE and STX. When such a transmission is received, the next output sent to the terminal occurs as a terminal transparent transmission. Nontransparent terminal output transmission resumes after the next nontransparent terminal input transmission.

Point-to-Point and Multidrop Line Access

The HASP TIP only supports nonaddressable HASP workstations on point-to-point communication lines.

Operating Mode

The HASP TIP is insensitive to line speeds.

The logical mode of operation of the TIP is half-duplex for each device. The TIP attempts to deliver console output whenever such data is available, unless an automatic input mode block has been delivered to the terminal.

The end of an input transmission block is signified by an ETB character code. This end-of-block indicator is part of the HASP protocol and cannot be changed.

Supported Input and Output Mechanisms

The TIP supports a range of input and output devices and mechanisms. Console devices have a keyboard as an input mechanism and a display as an output mechanism.

CDC-defined HASP batch devices are serviced concurrently with the console. All devices can be active at a given time, but only the input or output mechanism of the console device can be active at a given time. Input and output are therefore concurrent operations for batch devices and sequential operations for the console.

Card Reader Operation

Card reader operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Start input

Resume input

Abort input

The TIP sends command blocks to the host to:

Report when input stops

Report the end of the input stream

Report the number of cards read

Report the end of file

A HASP card reader becomes ready for input when the host sends the TIP a start input command block. Once card input begins, it continues until:

An end-of-file (EOF) block is received from the card reader

The host aborts input on the card reader stream

The host disconnects or disables the terminal

The workstation or communication line fails

Card input begins with the first card read. A card input file begins when the beginning-of-information is detected. The TIP expects a nonblank data card as the beginning-of-information initially or following an end of information card. The TIP discards blank cards, end-of-record cards, or end-of-information cards read when it is expecting the beginning of information.

When the beginning of information is located, the TIP converts subsequent data into a NOS operating system job file in 6-bit display code. Conversion from 026 or 029 punch patterns is performed.

The initial punch pattern translation for each job file is determined by the network administrator through the network configuration file. The terminal user can change the translation for subsequent portions of the job file by punching a 26 or a 29 in columns 79 and 80 of any job card or of any end-of-record card that is not in a transparent portion of the file.

The TIP converts nontransparent card input data from EBCDIC to 6-bit display code. Compressed characters are uncompressed.

Trailing blanks are discarded unless the last nonblank character is a colon; one blank is preserved or inserted when the last character is a colon or the card contains an odd number of significant characters. For hosts using a 64-character set, this convention produces an 82-character record when the character in column 80 is a colon.

Every card image becomes a zero-byte terminated record. These records become a network block equivalent to one, two, or three physical record units (PRUs); a block equivalent to one PRU contains 640 6-bit characters or equivalent zero-byte terminator bytes, left-justified in an 8-bit character byte. A record is continued in a subsequent network block when necessary. In the host, PIP removes the two low-order bits and packs the characters into actual PRUs.

Unless the TIP is reading transparent card data, it recognizes /*EOR cards as end-of-record indicators for NOS system logical records. It converts any numbers found in columns 6 and 7 to octal record level numbers. Level number 17 is recognized as an end-of-file indicator within a multiple file job. When necessary, short or zero-length PRUs are created.

Unless the TIP is reading transparent card data, it recognizes /*EOI cards as end-of-information indicators for NOS files. Once transparent input begins, the TIP recognizes only an end-of-file (EOF) block as end-of-information for the file.

Transparent card input (binary card data) is supported. The TIP does not translate subsequent portions of the job file when the user punches the characters TR in columns 79 and 80 of any end-of-record card. An end-of-record card containing such punches becomes the last recognized end-of-record card in the job file.

The TIP does not convert transparent input data to 6-bit display code. Transparent input characters are uncompressed and stored as 8-bit codes within 8-bit bytes. No record terminators are inserted. A record is continued in a subsequent network block when necessary.

These unterminated records become a network block equivalent to one, two, or three physical record units (PRUs). A block equivalent to one PRU contains 320 8-bit characters.

In the host, PIP packs the characters into actual PRUs. PIP right-justifies the 8-bit bytes received within 12-bit bytes for storage, zero-filling bits 8 through 11. Transparent card input becomes a variable number of variable length unterminated records containing 5 characters per word in each physical record unit of 64 60-bit words.

Printer Operation

Line printer operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report when a file limit is exceeded

Report the number of lines printed

Report a printer message (PM) in the output

Report the end of file

The TIP begins output to a printer when it has output queued for the device. Data received from the host is packed as line images in PRU-sized network blocks (the host PIP routine strips zero byte terminators and replaces them with hexadecimal FF code bytes). The TIP discards trailing blanks on a print line. End-of-record or end-of-file indicators in the data terminate an output transmission block.

6-bit display code is converted to EBCDIC. ASCII print files are converted from 7-bit codes right-justified in 8-bit bytes to the EBCDIC codes supported for normalized mode data (see Terminal Code Sets and Parity). This conversion includes stripping the unneeded bit 7. The host PIP routine removes the unneeded bits 8 through 11 of the stored 12-bit byte before transmission downline.

Printer busy and not-ready conditions are not reported to the host. Printing continues when the condition clears.

Once started, the TIP continues output to printers unless stopped or interrupted by a command block from the host or a printer message (PM carriage control sequence in the print file). Any printer that is interrupted or stopped can be restarted by a command block from the host.

The TIP supports both preprint and postprint printers. Different format effector substitutions occur, depending on the terminal class of the workstation. The TIP uses the subrecord control byte (SRCB) content appropriate to the terminal class.

Some HASP printers cannot suppress carriage control. These printers sometimes generate extra spaces in output and do not overprint lines.

Print Message Processing

When the print file contains a line beginning with the characters PM, the TIP stops the printer, notifies the application program through a command block, and sends the data from the print file line to the application program.

Card Punch Operation

Card punch operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report when a file limit is exceeded

Report the number of cards punched

Report the end of file

The TIP begins output to a punch when it has output queued for the device. Data received from the host is packed as card images in PRU-sized network blocks (the host PIP routine strips zero byte terminators and replaces them with hexadecimal FF code bytes).

End-of-record or end-of-file indicators in the data terminate an output transmission block. The TIP generates 7/8/9 multiple punch cards to mark end-of-record indicators, and /*EOI cards to mark end of file indicators.

6-bit display code is converted to EBCDIC. Punch pattern translation is selected by a command block preceding the data. If no such comand block is received, 029 punch patterns are assumed.

Punch busy and not-ready conditions are not reported to the host. Punching continues when the condition clears.

Once started, the TIP continues output to punches unless stopped or interrupted by a command block from the host. Any punch that is interrupted or stopped can be restarted by a command block from the host.

Plotter Operation

Plotter operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report when a file limit is exceeded

Report the number of plotter records delivered

Report the end of file

The TIP begins output to a plotter when it has output queued for the device. Data is received from the host in PRU-sized network blocks. End-of-record or end-of-file indicators in the data terminate an output transmission block.

Plotter data is always in network transparent mode. Plotter data consists of 6-bit or 8-bit binary codes and is therefore not converted to EBCDIC by CCP. PIP right-justifies 6-bit codes within the 8-bit character byte, with zero fill. The 8-bit codes in the host are stored as 5 characters per word, right-justified within 12-bit bytes with zero fill; PIP strips the unneeded bits 8 through 11 before transmission to CCP.

The data is formatted into HASP records by the TIP; normal data compression occurs. Each record ends when it contains the number of characters established as the page width of the device, or when the end of the network block is reached and the block is an end-of-record or end-of-information block. The initial page width is established by the network administrator in the network configuration file.

Plotter busy and not-ready conditions are not reported to the host. Plotting continues when the condition clears.

Once started, the TIP continues output to plotters unless stopped or interrupted by a command block from the host. Any plotter that is interrupted or stopped can be restarted by a command block from the host.

Normalized Mode Console Operation

The TIP supports only normalized mode input from a console, using only one input mode and mechanism: the keyboard in line mode. Input is forwarded after each console record is received (after each end-of-block indicator is recognized); available output is only sent after the end-of-block character.

The TIP supports only normalized mode output to a console, using only one output mechanism: the display. Line folding and scrolling is assumed to be performed by the output mechanism when page width (see PW command in appendix F) is reached. When output ends, the TIP ensures that the cursor is positioned at the lefthand margin of the next line on the display.

Transparent Mode

Network transparent mode input and output are only supported for batch devices. Network transparent input is allowed only from card readers. Network transparent output is allowed only to plotters, and is required for them.

The TIP does add the line protocol frame control characters to network transparent output and removes them from network transparent input. In network transparent mode, the application program is responsible for handling all data interpretation, including any embedded control characters.

Terminal Code Sets and Parity

The HASP TIP supports terminals operating without parity. HASP terminals use the IBM EBCDIC code set.

The TIP converts terminal code sets to and from network ASCII codes where necessary. During normalized output to HASP terminals, the TIP replaces ASCII characters that have no EBCDIC equivalents with blanks. During normalized input from HASP terminals, the TIP replaces EBCDIC characters that have no ASCII equivalents with blanks (see appendix A).

The card reader network input mode is initially nontransparent. The terminal user can change the mode to network transparent by using the characters TR on end-of-record cards in an input deck. This process is described earlier in this subsection under Card Reader Operation.

In network transparent mode, the TIP does not convert or delete characters in data.

The plotter output mode is fixed as network transparent. Plotter data can be either 6-bit instruction code bytes or 8-bit instruction code bytes. The TIP does not alter the byte content.

Normalized Mode and Network Nontransparent Mode Output

When the host sends ASCII character codes, the eighth upper bit (bit 7) can be anything and is ignored when the TIP translates the character. The bit is set as needed for the corresponding EBCDIC character.

Network Transparent Mode Output

The host can send anything for bit 7. Bit 7 is not changed by the CLA or the TIP. The entire 8-bit byte sent from the host is sent through the CLA to the terminal.

Normalized Mode and Network Nontransparent Mode Input

When the TIP sends the host ASCII character codes, the eighth bit is always set to 0.

Network Transparent Mode Input

The eighth bit (bit 7) of the data is not changed. All eight bits are sent to the host just as they came in on the communication line.

Initial Connection

The initial terminal class, stream numbers, configuration ordinal, and device types can be explicitly set through the network configuration file or can be determined by the TIP through automatic recognition. The need for automatic recognition is established through the network configuration file.

Automatic recognition is performed when a line becomes active. This process occurs as part of the initial communication sequence between the TIP and the terminal, as follows:

The terminal sends an enquiry block (ENQ).

The TIP responds with an acknowledgment block (ACK0).

The terminal sends a signon block on the console stream. The signon block contains either a /*SIGNON card image, or a /*CONFIG card image.

If the TIP receives a /*SIGNON card or an invalid /*CONFIG card, it discards the card and sends the diagnostic message

image <BAD CONFIG>

to the terminal, where image contains the portion of the card that cannot be processed.

If automatic recognition is required, the TIP waits for input of a valid /*CONFIG card from a card reader. The format of a valid /*CONFIG card is given in appendix H.

When a valid /*CONFIG card is received, the information is passed to the Communications Supervisor in the host. The Communications Supervisor compares it against the information known for all possible terminals using the line, until a match is found. The Communications Supervisor then reports the configuration to the TIP.

Once the signon block or /*CONFIG card is successfully processed, the terminal connection to the NPU is complete. Subsequent connection of the console and batch devices to a host and an application program depend on the network configuration; the procedure and possible commands required are described in appendix H. Batch devices are always connected to the host to which the console is connected.

If automatic recognition is not required for the communication line, any /*CONFIG card image received by the TIP is sent to the host as data. If the card image is received on the console stream, it is sent upline on the console connection. If the card image is received on a card reader stream, it is sent upline as the first card of a job file.

Disconnection

The HASP TIP does not recognize a /*SIGNOFF card as a method of disconnecting a terminal. A /*SIGNOFF card read as the only card in a job file is sent to the host as data.

The HASP TIP disconnects a terminal on a dial-up or a dedicated line when:

The terminal is not connected to a host and no input has been received from it for 2 minutes.

The Communications Supervisor requests that the line or terminal be disabled.

A line or terminal failure is detected.

When a console is disconnected from a host, the TIP disconnects all batch devices in the same terminal.

Data Formatting in Normalized Mode

During console input, the TIP detects the end of a physical line when:

An end-of-line indicator is detected; this end-of-line indicator is the end-of-record for the console stream.

During output to a console display device, a physical line ends when the output ends or when it contains a carriage return or unit separator code.

Normalized Editing Modes

Only one of the editing modes described in section 2 is supported: complete editing.

In complete editing mode, the cancel input character is stored as data and a flag is set in the network block. The flag becomes the can bit of the application block header in the host; refer to the CN command in appendix F. The TIP sends the message

DEL

to the terminal after the cancel input character is received.

Input Regulation

As mentioned earlier in this section under Data Priorities, the TIP must sometimes suspend input. The HASP protocol provides two levels of input regulation:

A global wait bit

Individual FCS stream bits

For devices assigned to a regulation level of priority 3 (normally batch devices), one or more of the bits in the FCS is set to zero and input is stopped between transmission blocks whenever a temporary suspension of input must occur. For devices assigned to a regulation level of priority 2 (normally console devices), only the wait bit is set; this action suspends all device input between transmission blocks.

Output from the host is delivered to the terminal during regulation. The FCS or the wait bits are changed and input resumes as soon as regulation ends and data can again be stored by the NPU.

Output Regulation

The terminal can also suspend output to a single device by using the FCS bits or to all devices by using the wait bit. The HASP TIP recognizes either use. When an FCS bit is used, the TIP presumes the wait bit is not used. When the wait bit is used, the TIP presumes the FCS bits are unchanged.

Error Recovery

The HASP TIP performs error recovery on input data by requesting retransmission of erroneous blocks from the terminal. The TIP detects the following errors:

Cyclic redundancy count errors

Illegal transmission block formats

Unknown responses

Timeout over the communication line

Block control byte (BCB) errors, such as a break in the sequence of transmitted blocks

When the TIP receives a negative acknowledgment block (NAK) that indicates an error in reception of downline data, the TIP attempts to retransmit the block 62 times. If the sixty-second attempt to transmit fails, the TIP classifies the communication line and terminal inoperative.

The TIP requests retransmission of a bad upline block 63 times. If the sixty-third attempt to receive fails, the TIP classifies the communication line and terminal inoperative.

Because EBCDIC does not contain a parity bit, the TIP cannot detect a parity error. It does not store erroneous code in the network block or set a flag for such data. The flag that becomes the pef bit in the host application block header is not used because data containing parity errors is discarded and replaced by correct data resent from the terminal.

BSC TIP

The BSC TIP supports IBM-2780- or IBM-3780-compatible bisynchronous terminals connected through dedicated or dial-up synchronous lines. Such terminals can contain devices that generate IVT data

to exchange with a host application program in an interactive mode, but normally contain devices that exchange PRU data with a host application program in a batch mode.

The TIP supports more than one terminal device per line. A 2780 or 3780 terminal always contains one card reader and one line printer; one card punch can also be supported.

The host software expects each terminal to have at least one interactive device, serviced through the interactive virtual terminal interface. The interactive device must have an input and an output mechanism. Because 2780 and 3780 terminals normally do not have consoles, the TIP simulates the existence of a console by using a special data mode for batch devices. The console simulation method is the reason each terminal must have one card reader and one line printer. The card reader is serviced as an interactive input mechanism and the line printer as an interactive output mechanism.

The terms terminal, terminal address, and device have varying or no definitions within 2780 or 3780 documentation. This subsection uses these definitions:

A terminal is a set of devices with a single point of access to a communication line.

A terminal address is the control character code used to identify a card punch transmission block.

A device is a separately accessible unit of input or output equipment, such as a console, a card reader, or a line printer. Within the network, a device is separately addressable, regardless of whether it has a separate hardware address within the terminal.

Terminal addresses are described in the Network Access Method Version 1 Network Definition Language Reference Manual, listed in the preface.

The following CDC-defined terminal classes are supported:

Terminal Class	Archetype Terminal
16	IBM 2780 remote job entry terminal
17	IBM 3780 remote job entry terminal

Site-defined devices in terminal classes 28 through 31 can also be supported if the site modifies the released version of the BSC TIP.

Each CDC-defined terminal class has a default set of terminal characteristics as shown in appendix F. These terminal characteristics can be changed from their defaults initially for each communication line through the network configuration file or subsequently by the terminal user and/or the application program through the use of commands.

The valid values and default parameters for commands in each of these terminal classes are shown in appendix F. When a valid value is entered from the console, the TIP responds with the message:

cd ACCEPTED..

where cd is the command mnemonic. When an invalid value is entered, the TIP responds with the message:

cd reason

The command represented by cd is ignored. The field identified as reason contains a message such as REJECTED, DUPLICATE CHARACTER, INAPPROPRIATE VALUE, ILLEGAL TERMINAL CLASS, ILLEGAL COMMAND, or ILLEGAL VALUE, explaining the reason the command was ignored.

The BSC TIP supports only the following IVT interface commands and features:

Changing the network control character (see the CT command in appendix F)

Changing the page width of the console (see the PW command in appendix F)

Displaying terminal characteristics (see the CH command in appendix F)

Sending a message to the network operator (see the MS command in appendix F)

Host node selection (see the HN command in appendix G)

Host availability display (see the HD command in appendix G)

Protocol Assumptions

This subsection describes those aspects of TIP operation that depend upon hardware use or are implementor-interpreted portions of 2780 or 3780 communications protocol. These aspects include:

Terminal feature support

Terminal transparent transmissions

Size and number of records per transmission block from or to a terminal

Resolving line contention

Line access

Operating mode

Terminal Features

The following 2780 terminal features are supported:

Transparent transmission format, separate from network transparent mode

Multiple record transmission blocks; from 2 to 7 records per block can be configured by the site administrator

120- and 144-character print lines

Batch job file termination when column 80 of the last card read contains an ETX code punch

Blank compression

The following 3780 terminal features are supported:

- Transparent transmission format, separate from network transparent mode
- Multiple record transmission blocks in transparent transmission format
- Punch component selection codes

Terminal Transparent Transmissions

Two forms of terminal input and output are supported:

- Terminal nontransparent transmissions
- Terminal transparent transmissions

In either form, the TIP recognizes two modes of data:

- Nontransparent network data
- Transparent network data

When a terminal begins communicating with the network, it uses nontransparent network data in nontransparent transmissions.

The BSC TIP accepts terminal transparent transmissions beginning with the sequence DLE and STX. When such a transmission is received, the next output sent to the terminal occurs as a terminal transparent transmission. Nontransparent terminal output transmission resumes after the next nontransparent terminal input transmission.

Record Blocking

A 2780 terminal using nontransparent transmissions sends or receives two (a default value, changeable by the network adminstrator through the network configuration file) 80-character fixed-length records or up to 7 variable-length records per block. A 3780 terminal using nontransparent transmissions sends or receives a maximum number of characters per transmission block, regardless of the number of records in the block. The number of 3780 characters used by the TIP during output also is initially established by the network administrator in the network configuration file; refer to the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface.

In transparent input transmissions, records always contain 80 characters. In nontransparent input transmissions, records contain from 1 through 80 characters.

Line Contention

Contention for line use is resolved in favor of input data. Batch output can be interrupted to accept card reader batch data or console data. Interrupted output is restarted when the end-of-transmission is detected in input. Card reader input is not interruptable.

Point-to-Point and Multidrop Line Access

Nonaddressable 2780 and 3780 terminals are supported on point-to-point communication lines only.

Operating Mode

The BSC TIP is insensitive to line speeds.

The logical mode of operation of the TIP is half-duplex for each device, with console input given priority over output. This means that console output is not started when input is active.

The TIP attempts to deliver console output whenever such data is available. Automatic input mode blocks are ignored by the BSC TIP.

Supported Input and Output Mechanisms

The TIP supports three batch input and output devices. Real console devices must emulate card reader input and accept line printer output if used.

Sequential Operation

For BSC terminals, only one input or one output mechanism can be active at any given time. Input and output are therefore sequential operations.

The TIP suspends console output when either the card reader becomes active or data is available for the line printer.

Batch input and output alternate unless one of the following events occurs:

- Output arrives for the console from the host; this output is printed as soon as the current batch print file ends.
- The card reader or printer is stopped (or suspended).
- Console input begins (the first two characters of a card are /* and the card reader hopper was previously empty).

Card Reader Operation

Card reader operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

- Start input
- Resume input
- Abort input

The TIP sends command blocks to the host to:

- Report when input stops
- Report the end of the input stream

Report the number of cards read

Report the end of file

A 2780 or 3780 card reader becomes ready for batch input when the host sends the TIP a start input command block. Card input begins with the first card read.

A card input file begins when the beginning-of-information is detected. The TIP expects a non-blank data card as the beginning-of-information initially or following an end of information card. The TIP discards blank cards, end-of-record cards, or end-of-information cards read when it is expecting the beginning of information.

When the beginning of information is located, the TIP converts subsequent data into a NOS operating system job file in 6-bit display code. Conversion from 026 or 029 punch patterns is performed.

The initial punch pattern translation for each job file is determined by the network administrator through the network configuration file. The terminal user can change the translation for subsequent portions of the job file by punching a 26 or a 29 in columns 79 and 80 of any job card or any end-of-record card that is not in a transparent portion of the file.

The TIP converts nontransparent card input data from EBCDIC to 6-bit display code. Compressed blanks and zeros are uncompressed.

Trailing blanks are discarded unless the last non-blank character is a colon; one blank is preserved or inserted when the last character is a colon or the card contains an odd number of significant characters. For hosts using a 64-character set, this convention produces an 82-character record when the character in column 80 is a colon.

Every card image becomes a zero-byte terminated record. These records become a network block equivalent to one, two, or three physical record units (PRUs); a block equivalent to one PRU contains 640 6-bit characters or equivalent zero-byte terminator bytes, left-justified in an 8-bit character byte. A record is continued in a subsequent network block when necessary. In the host, PIP removes the two low-order bits and packs the characters into actual PRUs.

Unless the TIP is reading network transparent card data or the terminal is sending terminal transparent transmissions, the TIP recognizes /*EOR cards as end-of-record indicators for NOS system logical records. It converts any numbers found in columns 6 and 7 to octal record level numbers. Record level 17 is recognized as an end-of-file indicator. When necessary, short or zero-length PRUs are created.

Unless the TIP is reading network transparent card data or the terminal is sending terminal transparent transmissions, the TIP recognizes /*EOI cards as end-of-information indicators for NOS files. If the terminal is sending transparent transmissions, the TIP also recognizes an ETX code punched in column 80 of the last card read (or in column 1 of an otherwise blank card) as an end-of-information indicator for a NOS file. Once transparent data input begins, the TIP recognizes only an end-of-transmission code (EOT) as an end-of-information for the file.

Network transparent card input (binary card data) is supported. The initial translation mode is always network nontransparent. The TIP does not translate subsequent portions of the job file when the user punches the characters TR in columns 79 and 80 of any end-of-record card. An end-of-record card containing such punches becomes the last recognized end-of-record card in the job file.

The TIP does not convert network transparent input data to 6-bit display code. Network transparent input characters are uncompressed and stored as 8-bit codes within 8-bit bytes. No record terminators are inserted. A record is continued in a subsequent network block when necessary.

These unterminated records become a network block equivalent to one, two, or three physical record units (PRUs). A block equivalent to one PRU contains 320 8-bit characters.

In the host, PIP packs the characters into actual PRUs. PIP right-justifies the 8-bit bytes received within 12-bit bytes for storage, zero-filling bits 8 through 11. Network transparent card input becomes a variable number of variable length unterminated records containing 5 characters per word in each physical record unit of 64 60-bit words.

Printer Operation

Line printer operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report when a file limit is exceeded

Report the number of lines printed

Report a printer message (PM) in the output

Report the end of file

The TIP begins output to a printer when it has output queued for the device and no input is in process. Data received from the host is packed as line images in PRU-sized network blocks (the host PIP routine strips zero byte terminators and replaces them with hexadecimal FF code bytes). The TIP discards trailing blanks on a print line. End-of-record or end-of-file indicators in the data terminate an output transmission block. The end of a file is also marked by sending an ETX block to the terminal.

6-bit display code is converted to EBCDIC. ASCII print files are converted from 7-bit codes right-justified in 8-bit bytes to the EBCDIC codes supported for normalized mode data (see Terminal Code Sets and Parity). This conversion includes stripping the unneeded bit 7. The host PIP routine strips the unneeded bits 8 through 11 from the 12-bit byte used for storage before sending the data downline.

Printer busy and not-ready conditions are not reported to the host. Printing continues when the condition clears.

Once started, the TIP continues output to printers unless stopped or interrupted by a command block from the host or a printer message (PM carriage control sequence in the print file). Any printer that is interrupted or stopped can be restarted by a command block from the host.

Print Message Processing

When the print file contains a line beginning with the characters PM, the TIP stops the printer, notifies the application program through a command block, and sends the data from the print file line to the application program.

Card Punch Operation

Card punch operations are governed by special command blocks exchanged with the host. These commands originate within the PIP routine or are coverted by PIP to or from supervisory messages exchanged with an application program. The host sends commands to the TIP to:

Terminate output

Mark the point of output termination

Restart output

The TIP sends command blocks to the host to:

Report when a file limit is exceeded

Report the number of cards punched

Report the end of file

The TIP begins output to a punch when it has output queued for the device. Data received from the host is packed as card images in PRU-sized network blocks (the host PIP routine strips zero byte terminators and replaces them with hexadecimal FF code bytes).

End-of-record or end-of-file indicators in the data terminate an output transmission block. The TIP generates /*EOR cards to mark end-of-record indicators, and /*EOI cards to mark end-of-file indicators. The end of a file is also marked by sending an ETX block to the terminal.

6-bit display code is converted to EBCDIC. Punch pattern translation is selected through a command block that precedes the data. If no command block is received, 029 punch patterns are assumed.

The TIP prefixes the first card image in each punch file with the terminal address code associated with the punch. For 3780 terminals, this code is initially established by the network administrator in the network configuration file; refer to the Network Access Method Version 1 Network Definition Language Reference Manual listed in the preface. The code used for a 3780 can also be established by the terminal user during the automatic recognition process, described later in this subsection.

Punch busy and not-ready conditions are not reported to the host. Punching continues when the condition clears.

Once started, the TIP continues output to punches unless stopped or interrupted by a command block from the host. Any punch that is interrupted or stopped can be restarted by a command block from the host.

Normalized Mode Console Operations

The TIP supports only normalized mode input from a console, using only only one input mode and mechanism: the card reader, emulating a keyboard in line mode. Console input begins with the characters

```
/*
```

in character positions 1 and 2 of a card image. Input is forwarded after the card ends and the end-of-transmission character is recognized; available output is only sent after the end-of-transmission character.

Console output is sent only after output of a batch file is completed. The first line of any console output is prefixed by the message described in section 2.

The TIP supports only normalized mode input from a console, using only one output mechanism: the line printer, emulating a display. Line folding and paging is performed using carriage control characters when page width (see PW command in appendix F) is reached.

Network Transparent Mode

Network transparent mode input is only supported for the card reader. The TIP does remove the line protocol frame control characters from input.

In network transparent mode, the application program is responsible for handling all data interpretation, including any embedded control characters.

Terminal Code Sets and Parity

The BSC TIP supports terminals operating without parity. 2780 and 3780 terminals use the IBM EBCDIC code set.

The TIP converts terminal code sets to and from network ASCII codes where necessary. During normalized output to BSC terminals, the TIP replaces ASCII characters that have no EBCDIC equivalents with blanks. During normalized input from BSC terminals, the TIP replaces EBCDIC characters that have no ASCII equivalents with blanks (see appendix A).

In network transparent mode, the TIP does not convert or delete characters in data.

The card reader input mode is initially network nontransparent. The terminal user can change the mode to network transparent by using the characters TR on certain cards in an input deck. This process is described earlier in this subsection under Card Reader Operation.

Normalized Mode and Network Nontransparent Mode Output

When the host sends ASCII character codes, the eighth bit (bit 7) can be anything and is ignored when the TIP translates the character. The bit is set as needed for the corresponding EBCDIC character.

Network Transparent Mode Output

The host can send anything for bit 7. Bit 7 is not changed by the CLA or the TIP. The entire 8-bit byte sent from the host is sent through the CLA to the terminal.

Normalized Mode and Network Nontransparent Mode Input

When the TIP sends the host ASCII character codes, the eighth bit is always set to 0.

Network Transparent Mode Input

The eighth bit (bit 7) is not changed. All eight bits are sent to the host just as they came in on the communication line.

Initial Connection

The initial terminal class, terminal address code for the punch, configuration ordinal, and device types can be explicitly set through the network configuration file or can be determined by the TIP through automatic recognition. The need for automatic recognition is established through the network configuration file.

Automatic recognition is performed when a line becomes active. This process occurs as part of the initial communication sequence between the TIP and the terminal, as follows:

The terminal sends an enquiry block (ENQ).

The TIP responds with an acknowledgment block (ACK0).

The terminal sends a /*CONFIG card image.

If the TIP receives an invalid /*CONFIG card, it discards the card and sends the diagnostic message

image <BAD CONFIG>

to the terminal, where image contains the portion of the card that cannot be processed.

If automatic recognition is required, the TIP waits for input of a valid /*CONFIG card from a card reader. The format of a valid /*CONFIG card is given in appendix H.

When a valid /*CONFIG card is received, the information is passed to the Communications Supervisor in the host. The Communications Supervisor compares it against the information known for all possible terminals using the line, until a match is found. The Communications Supervisor then reports the configuration to the TIP.

Once the /*CONFIG card is successfully processed, the terminal connection to the NPU is complete. Until the processing is completed, the TIP discards any input received from the terminal. Subsequent connection of the console and batch devices to a host and an application program depend on the network configuration; the procedure and possible commands required are described in appendix H. Batch devices are always connected to an application program in the host to which the console is connected.

If automatic recognition is not required for the communication line, any /*CONFIG card image received by the TIP is sent to the host as data. Because the card image is received on the console stream, it is sent upline on the console connection.

Disconnection

The BSC TIP disconnects a terminal on a dial-up line when:

The terminal uses automatic recognition information not recognized by the TIP.

The terminal user fails to complete the automatic recognition sequence within the time allowed.

The terminal is not connected to a host and no input has been received from it for 2 minutes.

The Communications Supervisor requests that the line or terminal be disabled.

The BSC TIP does not disconnect a terminal on a dedicated line. Input from disabled terminals or input on disabled hardwired lines is discarded. Failure to complete automatic recognition on a dedication line causes the TIP to restart the procedure.

Data Formatting in Normalized Mode

During input from the emulated console, the TIP detects the end of a physical line when a card image ends and the end-of-transmission code is received. During output to the emulated console device with a display mechanism, a physical line ends when the output ends, when the page width is reached, or when the output contains a carriage return or line feed code.

Normalized Editing Modes

Only one of the editing modes described in section 2 is supported: complete editing.

Input Regulation

As mentioned earlier in this section under Data Priorities, the TIP must sometimes suspend input. If input is in process when regulation begins, the TIP sends a wait/acknowledgment message (WACK) to the terminal; this action suspends the input between transmission blocks whenever a temporary suspension of input must occur. The data transmitted from the terminal is not discarded. If the terminal is idle when regulation begins, the TIP prevents subsequent input by sending an end-of-transmission (EOT) code to the terminal.

Output from the host is sent to the terminal during regulation. When regulation ends and data can again be stored by the NPU, the TIP sends an enquiry block (ENQ) to resume input at the point where it was suspended.

Error Recovery

The BSC TIP performs error recovery on input data by requesting retransmission of erroneous blocks from the terminal. The TIP detects the following errors:

Cyclic redundancy count errors

Illegal transmission block formats

Unknown responses

Timeout over the communication line

When the TIP receives a negative acknowledgment block (NAK) that indicates an error in reception of downline data, the TIP attempts to retransmit the block three times. If the fourth attempt to transmit fails, the TIP classifies the communication line and terminal inoperative.

The TIP requests retransmission of a bad upline block three times. If the fourth attempt to receive fails, the TIP classifies the communication line and terminal inoperative.

Because EBCDIC does not contain a parity bit, the TIP cannot detect a parity error. The flag that becomes the pef bit in the host application block header is not used because data containing parity errors is discarded and replaced by correct data resent from the terminal.

NETWORK FAILURE AND RECOVERY 8

This section describes the types of network failure that are possible. Each type of failure requires its own recovery techniques.

APPLICATION PROGRAMS

The present release of the network software makes no provision for data recovery if NIP or NVF failure occurs. The operator must reinitiate NAM. All application programs that are not system control point jobs are aborted. When the network processing unit detects a network communication failure, it indicates the condition by displaying a message on all connected terminals.

If the Network Access Method fails (specifically, if NIP communication fails), the network software dumps NAM and enters a message in the system dayfile. All application programs that are not system control point jobs are aborted, and a message is issued to the dayfile of each job.

An aborted application program can reprieve itself under certain conditions without being reloaded. These conditions are described in section 6 and appendix B. A reprieved application program must issue a NETOFF call before it can issue a new NETON call. A new NETON call can be successfully completed as soon as a copy of the Network Access Method is restarted. If the reprieved program issues the NETOFF after the Network Access Method is restarted, the NETOFF is ignored.

HOST

If a host fails, the Network Processing Unit (NPU) and its software must stop message processing to that host. Host unavailability is communicated to the other ends of all logical links. Also, the NPU sends an informative service message to all connected, interactive terminals (and to some other types of terminals) informing the terminal that the host is unavailable. After recovery, all logical links are reinitialized and new connections are made.

The host recovers the existing configuration status by means of status requests to the NPU.

NETWORK PROCESSING UNIT

If an NPU fails, it must be reloaded from the host. Off-line diagnostic tests may be desirable during this period to help identify the cause of failure. Failure is detected by means of a 20-second timeout across the coupler. The NPU is forced to generate a load request message.

An NPU that has failed can be dumped before it is reloaded. Whenever an NPU fails, it is automatically reloaded by the Network Supervisor (NS). When the NPU is reloaded, it requests supervision from the Communications Supervisor (CS). CS then informs the NPU Operator and the Host Operator that it is now supervising the NPU.

LOGICAL LINK

Host failure, one of the causes of link failure, was previously described. Link protocol failure leads to regulation of data traffic until all message traffic ceases on the link.

A logical link may recover spontaneously (regulation level drops), or may be reinitialized by the host. In the case of spontaneous recovery, the logical link protocol allows a restart without loss of data. Otherwise, all logical connections must be remade. Trunks connecting neighboring NPUs are a special class of links. Trunk recovery protocol is handled by the Link Interface Package (LIP).

TRUNK

A trunk failure is detected by a failure of the trunk protocol. All data queued for transmission on the trunk is discarded. The failure is reported to the host. The trunk protocol detects the trunk recovery. The logical link protocol determines when the trunk can again be used for data block transmissions.

LINE

Lines are disconnected and terminal control blocks (TCBs) associated with the lines are deleted. A line failure is detected by abnormal modem status or by the line protocol failure. The change of status is reported by CCP to NAM in the host.

The line is constantly monitered by CCP, and if the correct modem signals are present, CCP reactivates the line and requests TCB configuration from CS.

TERMINAL

Terminal status is reported and messages are discarded. TCBs are not released. Once terminal failure has been detected, possible terminal recovery is monitored by a periodic status check or diagnostic poll made from the NPU to the terminal. Terminal recovery status is reported to NAM.

CHARACTER DATA INPUT, OUTPUT, AND CENTRAL MEMORY REPRESENTATION

A

This appendix describes the code and character sets used by the operating system local batch device driver programs, magnetic tape driver programs, and network terminal communication products. This appendix does not describe how other products associate certain graphic or control characters with specific binary code values for collating or syntax processing purposes. The main text of this manual describes such associations that are relevant to the reader.

CHARACTER SETS AND CODE SETS

A character set differs from a code set. A character set is a set of graphic and/or control character symbols. A code set is a numbering system used to represent each character within a character set. Characters exist outside the computer system and communication network; codes are received, stored, retrieved, and transmitted within the computer system and network.

When this manual refers to the ASCII 128-character set or the 7-bit ASCII code set, it is referring to the character set and code set defined as the American National Standard Code for Information Interchange (ASCII, ANSI Standard X3.4-1977). References in this manual to an ASCII character set or an ASCII code set do not necessarily apply to the 128-character, 7-bit ASCII code set.

GRAPHIC AND CONTROL CHARACTERS

A graphic character can be displayed or printed. Examples of graphic characters are the characters A through Z, a blank, and the digits 0 through 9. A control character is not a graphic character; a control character initiates, modifies, or stops a control operation. An example of a control character is the backspace character, which moves the terminal carriage or cursor back one space. Although a control character is not a graphic character, some terminals use a graphic representation for control characters.

CODED AND BINARY CHARACTER DATA

Character codes can be interpreted as coded character data or as binary character data. Coded character data is converted by default from one code set representation to another as it enters or leaves the computer system; for example, data received from a terminal or sent to a magnetic tape unit is converted. Binary character data is not converted as it enters or leaves the system. Character codes are not converted when moved within the system; for example, data transferred to or from mass storage is not converted.

The distinction between coded character data and binary character data is important when reading or punching cards and when reading or writing magnetic tape. Only coded character data can be properly reproduced as characters on a line printer. Only binary character data can properly represent characters on a punched card when the data cannot be stored as display code.

The distinction between binary character data and characters represented by binary data (such as peripheral equipment instruction codes) is also important. Only binary noncharacter data can properly reproduce characters on a plotter.

CHARACTER SET TABLES

The character set tables in this appendix are designed so that the user can find the character represented by a code (such as in a dump) or find the code that represents a character. To find the character represented by a code, the user looks up the code in the column listing the appropriate code set and then finds the character on that line in the column listing the appropriate character set. To find the code that represents a character, the user looks up the character and then finds the code on the same line in the appropriate column.

NETWORK OPERATING SYSTEM

NOS supports the following character sets:

- CDC graphic 64-character set
- CDC graphic 63-character set
- ASCII graphic 64-character set
- ASCII graphic 63-character set
- ASCII graphic 95-character set
- ASCII 128-character set

Each installation must select either a 64-character set or a 63-character set. The differences between the codes of a 63-character set and the codes of a 64-character set are described under Character Set Anomalies. Any reference in this appendix to a 64-character set implies either a 63- or 64-character set unless otherwise stated.

NOS supports the following code sets to represent its character sets in central memory:

- 6-bit display code
- 12-bit ASCII code
- 6/12-bit display code

The 6-bit display code is a set of octal codes from 00 to 77, inclusive.

The 12-bit ASCII code is the ASCII 7-bit code right-justified in a 12-bit byte. The bits are numbered from the right starting with 0; bits 0 through 6 contain the ASCII code, bits 7 through 10 contain zeros, and bit 11 distinguishes the 12-bit ASCII 0000 code from the 12-bit 0000 end-of-line byte. The octal values for the 12-bit codes are 0001 through 0177 and 4000.

The 6/12-bit display code is a combination of 6-bit codes and 12-bit codes. The octal values for the 6-bit codes are 00 through 77, excluding 74 and 76. (The interpretation of the 00 and 63 codes is described under Character Set Anomalies in this appendix.) The octal 12-bit codes begin with either 74 or 76 and are followed by a 6-bit code. Thus, 74 and 76 are escape codes and are never used as 6-bit codes within the 6/12-bit display code set. The octal values of the 12-bit codes are: 7401, 7402, 7404, 7407, and 7601 through 7677. The other 12-bit codes, 74xx and 7600, are undefined.

CHARACTER SET ANOMALIES

The operating system input/output software and some products interpret two codes differently when the installation selects a 63-character set rather than a 64-character set. If a site uses a 63-character set: the colon (:) graphic character is always represented by a 6-bit display code value of 63 octal; display code 00 is undefined (it has no associated graphic or punched card code); the percent (%) graphic does not exist, and translations produce a space (55 octal).

However, if the site uses a 64-character set, output of a an octal 7404 6/12-bit display code or a 6-bit display code value of 00 produces a colon. In ASCII mode, a colon can be input only as a 7404 6/12-bit display code. Undefined 6/12-bit display codes in output files produce unpredictable results and should be avoided.

Two consecutive 6-bit display code values of 00 can be confused with the 12-bit 0000 end-of-line byte and should be avoided.

INTERACTIVE TERMINAL USERS

NOS supports display consoles and teletypewriters that use code sets other than 7-bit ASCII codes for communication or use graphics other than those defined in an ASCII character set. Data exchanged with such terminals is translated as described under Terminal Transmission Modes in this appendix. The following description applies only to terminals that use 7-bit ASCII codes and the ASCII character set.

ASCII Data Exchange Modes

Table A-1 shows the character sets and code sets available to an Interactive Facility (IAF) user. Table A-2 shows the octal and hexadecimal 7-bit ASCII code for each ASCII character, and can be used to convert codes from octal to hexadecimal. (Certain Terminal Interface Program commands require hexadecimal specification of a 7-bit ASCII code.)

IAF supports both normalized mode and transparent mode transmissions through the network. These transmission modes are described under Terminal Transmission Modes in this appendix. Refer to the NOS Version 2 Reference Set, Volume 3 System Commands, for additional information.

IAF treats normalized mode transmissions as coded character data; IAF converts these transmissions to or from either 6-bit or 6/12-bit display code.

IAF treats transparent mode transmissions as binary character data. Transparent mode input or output uses 12-bit bytes, with bit 11 always set to 1; for ASCII terminals, transparent mode input and output occurs in the 12-bit ASCII code shown in table A-1, but the leftmost digit is 4 instead of 0.

When the NORMAL command is in effect, IAF assumes that the ASCII graphic 64-character set is used and translates all input and output to or from display code. When the ASCII command is in effect, IAF assumes that the ASCII 128-character set is used and translates all input and output to or from 6/12-bit display code.

The IAF user can convert a 6/12-bit display code file to a 12-bit ASCII code file using the NOS FCOPY control statement. The resulting 12-bit ASCII file can be routed to a line printer but the file cannot be output through IAF.

Terminal Transmission Modes

Coded character data can be exchanged with a conversational terminal in two transmission modes. These two modes, normalized mode and transparent mode, correspond to the types of character code editing and translation performed by the network software during input and output operations.

The terminal operator can change the input transmission mode using a terminal definition command (sometimes called a Terminal Interface Program command). The application program providing the terminal facility service can change the input or output transmission mode.

Normalized Mode Transmissions

Normalized mode is the initial and default mode used for both input and output transmissions. The network software translates normalized mode data to or from the transmission code used by the terminal into or from the 7-bit ASCII code shown in table A-2. (Tables A-1 and A-3 through A-7 are provided for use while coding an application program to run under the operating system; they do not describe character transmissions through the network.) Translation of a specific terminal transmission code to or from a specific 7-bit ASCII code depends on the terminal class in which the network software places the terminal.

The following paragraphs summarize the general case for normalized mode data code translations. This generalized description uses table A-2.

The reader can extend this generalized description by using the other tables to determine character set mapping for functions initiated from a terminal. For example, the description under Terminal Output Character Sets can be used to predict whether a lowercase ASCII character stored in 6/12-bit display code can appear on an EBCDIC or external BCD terminal; if an ASCII character passes through the network represented in 7-bit ASCII as normalized mode data, it probably can be represented on an EBCDIC terminal, but it is always transformed to an uppercase character on a mode 4A ASCII terminal.

Table A-2 contains the ASCII 128-character set supported by the network software. The ASCII 96-character subset in the rightmost six columns minus the deletion character (DEL) comprises the graphic 95-character subset; the DEL is not a graphic character, although some terminals graphically represent it. The graphic 64-character subset comprises the middle four columns. Only the characters in this 64-character subset have 6-bit display code equivalents.

Terminals that support an ASCII graphic 64-character subset actually use a subset of up to 96 characters, consisting of the graphic 64-character subset and the control characters of columns 1 and 2; often, the DEL character in column 7 is included. Terminals that support an ASCII graphic 95-character or 96-character subset actually might use all 128 characters.

The hexadecimal value of the 7-bit code for each character in table A-2 consists of the character's column number in the table, followed by its row number. For example, N is in row E of column 4, so its hexadecimal value is 4E. The octal value for the code when it is right-justified in an 8-bit byte appears beneath the character graphic or mnemonic. The binary value of the code consists of the bit values shown, placed in the order given by the subscripts for the letter b; for example, N is 1001110_b.

Tables A-8 through A-19 show the normalized mode translations performed for each terminal class. The parity shown in the terminal transmission codes is the parity used as a default for the terminal class. The parity setting actually used by a terminal can be identified to the network software through a TIP command.

Tables A-8 through A-19 contain the graphic and control characters associated with the transmission codes used by the terminal because of the terminal class and code set in use. The network ASCII graphic and control characters shown are those of the standard ASCII character set associated with the ASCII transmission codes of table A-2.

Terminal Output Character Subsets -- Although the network supports the ASCII 128-character set, some terminals restrict output to a smaller character set. This restriction is supported by replacing the control characters in columns 0 and 1 of table A-2 with blanks to produce the ASCII graphic 95-character subset, and replacing the characters in columns 6 and 7 with the corresponding characters from columns 4 and 5, respectively, to produce the ASCII graphic 64-character subset.

Terminal Input Character Subsets and Supersets -- Although the network supports the ASCII 128-character set, some terminals restrict input to a smaller character set or permit input of a larger character set. A character input from a device using a character set other than ASCII is converted to an equivalent ASCII character; terminal characters without ASCII character equivalents are represented by the ASCII code for a space.

Site-written terminal-servicing facility programs can also cause input or output character replacement, conversion, or deletion by exchanging data with the network in 6-bit display code.

Input Restrictions -- The network software automatically deletes codes associated with terminal communication protocols or terminal hardware functions. These codes usually represent the cancel, backspace, linefeed, carriage return, and deletion characters. If input device control or paper tape support is requested, the device control 3 and device control 1 codes are also deleted. Some of these code deletions can be suppressed by using editing options (refer to the FA and SE terminal definition parameters in the NOS Version 2 Reference Set, Volume 3, System Commands).

Output Restrictions -- All codes sent by an application program are transmitted to the terminal. However, the 12-bit ASCII code 0037 (octal), the 6/12-bit display code 7677 (octal), and the 7-bit ASCII code 1F (hexadecimal) should be avoided in normalized mode output. The network software interprets the unit separator character represented by these codes as an end-of-line indicator. The processing of application program-supplied unit separators causes incorrect formatting of output and can cause loss of other output characters.

Input Parity Processing -- The network software does not preserve the parity of the terminal transmission code in the corresponding ASCII code. An ASCII code received by the terminal-servicing facility program always contains zero as its eighth bit.

Output Parity Processing -- The network software provides the parity bit setting appropriate for the terminal being serviced, even when the software is translating from ASCII character codes with zero parity bit settings.

Transparent Mode Transmissions

Transparent mode is selected separately for input and output transmissions.

During transparent mode input, the parity bit is stripped from each terminal transmission code (unless the N parity option has been selected by a terminal definition command), and the transmission code is placed in an 8-bit byte without translation to 7-bit ASCII code. Line transmission protocol characters are deleted from mode 4 terminal input. When the 8-bit bytes arrive in the host computer, a terminal servicing facility program can right-justify the bytes within a 12-bit byte.

During transparent mode output, processing similar to that performed for input occurs. When the host computer transmits 12-bit bytes, the leftmost 4 bits (bits 11 through 8) are discarded. The code in each 8-bit byte received by the network software is not translated. The parity bit appropriate for the terminal class is altered as indicated by the parity option in effect for the terminal. The codes are then transmitted to the terminal in bytes of a length appropriate for the terminal class. Line transmission protocol characters are inserted into mode 4 terminal output.

LOCAL BATCH USERS

Table A-3 lists the CDC graphic 64-character set, the ASCII graphic 64-character set, and the ASCII graphic 95-character set available on local batch devices. This table also lists the code sets and card keypunch codes (026 and 029) that represent the characters.

The 64-character sets use 6-bit display code as their code set; the 95-character set uses 12-bit ASCII code. The 95-character set is composed of all the characters in the ASCII 128-character set that can be printed at a line printer (refer to Line Printer Output). Only 12-bit ASCII code files can be printed using the graphic ASCII 95-character set. The 95-character set is represented by the octal 12-bit ASCII codes 0040 through 0176. An octal 12-bit ASCII code outside of the range 0040 through 0176 represents an unprintable character.

To print a 6/12-bit display code file, the user must convert the file to 12-bit ASCII code. The NOS FCOPY control statement is used for this conversion.

Line Printer Output

The printer train used on the line printer to which a file is sent determines which batch character set is printed. The following CDC print trains match the batch character sets in table A-3:

Character Set	Print Train
CDC graphic 64-character set	596-1
ASCII graphic 64-character set	596-5
ASCII graphic 95-character set	596-6

The characters of the default 596-1 print train are listed in the table A-3 column labeled CDC Graphic (64-Character Set); the 596-5 print train characters are listed in the table A-3 column labeled ASCII Graphic (64-Character Set); and the 596-6 print train characters are listed in the table A-3 column labeled ASCII Graphic (95-Character Set).

If an unprintable character exists in a line, NOS marks the condition by printing the number sign (#) in the first printable column of the line. A space replaces the unprintable character within the line.

When a transmission error occurs during the printing of a line, NOS makes up to five attempts to reprint the line. The CDC graphic print train prints a concatenation symbol (↱) in the first printable column of a line containing errors. The ASCII print trains print an underline (_) instead of the concatenation symbol.

After the fifth attempt, the setting of sense switch one for the batch input and output control point determines further processing. NOS either rewinds the file and returns it to the print queue, or ignores the transmission errors.

Punched Card Input and Output

A character represented by multiple punches in a single column has its punch pattern identified by numbers and hyphens. For example, the punches representing an exclamation point are identified as 11-0; this notation means both rows 11 and 0 are punched in the same column.

A multiple punch pattern that represents something other than a character is identified by numbers and slashes. For example, the punches representing the end of an input file are identified as 6/7/8/9; this notation means rows 6 through 9 are punched in the same column.

Coded character data is exchanged with card readers or card punches according to the translations shown in table A-3. As indicated in the table, other card keypunch codes are available for input of the ASCII and CDC characters [and]. NOS cannot read or punch the 95-character set as coded character data.

Each site chooses either 026 or 029 as its default keypunch code. NOS begins reading an input deck in the default code (regardless of the character set in use). The user can specify the alternate keypunch code by punching a 26 or 29 in columns 79 and 80 of any job card, 6/7/9 card, or 7/8/9 card. The specified translation continues throughout the job unless the alternate keypunch code translation is specified on a subsequent 6/7/9 or 7/8/9 card.

A 5/7/9 card with a punch in column 1 changes keypunch code translation if the card is read immediately before or after a 7/8/9 card. A space (no punch) in column 2 indicates 026 translation mode; a 9 punch in column 2 indicates 029 translation mode. The specified translation remains in effect until a similar 5/7/9 card or a 7/8/9 card is encountered, or the job ends.

The 5/7/9 card also allows literal input when 4/5/6/7/8/9 is punched in column 2. Literal input can be used to read 80-column binary character data within a punched card deck of coded character data.

Literal cards are stored with each column represented in a 12-bit byte (a row 12 punch is represented by a 1 in bit 11, row 11 by a 1 in bit 10, row 0 by a 1 in bit 9, and rows 1 through 9 by 1's in bits 8 through 0 of the byte), using 16 central memory words per card. Literal input cards are read until another 5/7/9 card with 4/5/6/7/8/9 punched in column 2 is read. The next card can specify a new conversion mode.

If the card following the 5/7/9, 6/7/9, or 7/8/9 card has a 7 and a 9 punched in column 1, the section of the job deck following it contains system binary cards (as described in the NOS Version 2 Reference Set, Volume 3, System Commands).

REMOTE BATCH USERS

Remote batch console input and output is restricted to normalized mode transmission. Normalized mode is described under Terminal Transmission Modes in this appendix.

The abilities to select alternate keypunch code translations, to read binary cards, to output plotter files, and to print lowercase characters depend upon the remote terminal equipment. Remote batch terminal support under NOS is described in the Remote Batch Facility (RBF) reference manual.

MAGNETIC TAPE USERS

The character and code sets used for reading and writing magnetic tapes depend on whether coded or binary data is read or written and on whether the tape is 7-track or 9-track.

Coded Data Exchanges

Coded character data to be copied from mass storage to magnetic tape is assumed to be stored in a 63- or 64-character 6-bit display code. The operating system magnetic tape driver program converts the data to 6-bit external BCD code when writing a coded 7-track tape and to 7-bit ASCII or 8-bit EBCDIC code (as specified on the tape assignment statement) when writing a coded 9-track tape.

Coded character data copied to mass storage from magnetic tape is stored in a 63- or 64-character 6-bit display code. The operating system magnetic tape driver program converts the data from 6-bit external BCD code when reading a coded 7-track tape and from 7-bit ASCII or 8-bit EBCDIC code (as specified on the tape assignment statement) when reading a coded 9-track tape.

To read and write lowercase character 7-bit ASCII or 8-bit EBCDIC codes or to read and write control codes, the user must assign a 7-track or 9-track tape in binary mode.

Seven-Track Tape Input and Output

Table A-4 shows the code and character set conversions between 6-bit external BCD and 6-bit display code for 7-track tapes. Because only 63 characters can be represented in 7-track even parity, one of the 64 display codes is lost in conversion to and from external BCD code.

Figure A-1 shows the differences in 7-track tape conversion that depend on whether the system uses the 63-character or 64-character set. The ASCII character for the specified character code is shown in parentheses. The output arrows show how the 6-bit display code changes when it is written on tape in external BCD. The input arrows show how the external BCD code changes when the tape is read and converted to display code.

63-Character Set

Display Code		External BCD		Display Code
00		16(%)		00
33(0)	Output →	12(0)	Input →	33(0)
63(:)		12(0)		33(0)

64-Character Set

Display Code		External BCD		Display Code
00(:)		12(0)		33(0)
33(0)	Output →	12(0)	Input →	33(0)
63(%)		16(%)		63(%)

Figure A-1. Magnetic Tape Code Conversions

Nine-Track Tape Input and Output

Table A-5 lists the conversions between the 7-bit ASCII code used on the tape and the 6-bit display code used within the system. Table A-6 lists the conversions between the 8-bit EBCDIC code used on the tape and the 6-bit display code used within the system.

When an ASCII or EBCDIC code representing a lowercase character is read from a 9-track magnetic tape, it is converted to its uppercase character 6-bit display code equivalent. Any EBCDIC code not listed in table A-6 is converted to display code 55 (octal) and becomes a space. Any code between 80 (hexadecimal) and FF (hexadecimal) read from an ASCII tape is converted to display code 00.

Binary Character Data Exchanges

Binary character data exchanged between central memory files and magnetic tape is transferred as a string of bytes without conversion of the byte contents. The grouping of the bytes and the number of bits in each byte depend on whether 7-track or 9-track tape is being used.

Seven-Track Tape Input and Output

Each binary data character code written to or read from 7-track magnetic tape is assumed to be stored in a 6-bit byte, such as the system uses for 63- or 64-character 6-bit display code. Seven-bit ASCII and 8-bit EBCDIC codes can only be read from or written to 7-track magnetic tape as binary character data if each code is stored within a 12-bit byte as if it were two character codes.

Nine-Track Tape Input and Output

Each binary data character code exchanged between central memory files and 9-track magnetic tape is assumed to be stored in an 8-bit or 12-bit byte. During such binary data transfers, the 6/12-bit

display codes and 12-bit ASCII codes shown in table A-1, the 7-bit ASCII codes shown in table A-2, or or the 8-bit hexadecimal EBCDIC codes shown in table A-7 can be read or written. The 7-bit ASCII codes and 8-bit EBCDIC codes can be exchanged either in an unformatted form or right-justified within a zero-filled 12-bit byte of memory.

When 9-track tape is written, every pair of 12-bit memory bytes becomes three 8-bit tape bytes; when 9-track tape is read, every three 8-bit tape bytes become a pair of 12-bit memory bytes. Because of the 12-bit byte pairs, codes not packed into 12-bit bytes are exchanged in their unpacked form, while codes packed in 12-bit bytes are exchanged in packed form.

When an odd number of central memory words is read or written, the lower four bits of the last 8-bit byte (bits 0 through 3 of the last word) are not used. For example, three central memory words are written on tape as 22 8-bit bytes (7.5 pairs of 12-bit bytes) and the remaining four bits are ignored.

CODE CONVERSION AIDS

Table A-7 contains the octal values of each 8-bit EBCDIC code right-justified in a 12-bit byte with zero fill. This 12-bit EBCDIC code can be produced or read using the FORM and 8-Bit Subroutines utilities.

TABLE A-1. INTERACTIVE TERMINAL CHARACTER SETS

Character Sets		Code Sets		
ASCII Graphic (64-Character Set)	ASCII Character (128-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code
: colon††		00††		
A	A	01	01	0101
B	B	02	02	0102
C	C	03	03	0103
D	D	04	04	0104
E	E	05	05	0105
F	F	06	06	0106
G	G	07	07	0107
H	H	10	10	0110
I	I	11	11	0111
J	J	12	12	0112
K	K	13	13	0113
L	L	14	14	0114
M	M	15	15	0115
N	N	16	16	0116
O	O	17	17	0117
P	P	20	20	0120
Q	Q	21	21	0121
R	R	22	22	0122
S	S	23	23	0123
T	T	24	24	0124
U	U	25	25	0125
V	V	26	26	0126
W	W	27	27	0127
X	X	30	30	0130
Y	Y	31	31	0131
Z	Z	32	32	0132
0	0	33	33	0060
1	1	34	34	0061
2	2	35	35	0062
3	3	36	36	0063
4	4	37	37	0064
5	5	40	40	0065
6	6	41	41	0066
7	7	42	42	0067
8	8	43	43	0070
9	9	44	44	0071
+ plus	+ plus	45	45	0053
- hyphen (minus)	- hyphen (minus)	46	46	0055
* asterisk	* asterisk	47	47	0052
/ slant	/ slant	50	50	0057
(opening parenthesis	(opening parenthesis	51	51	0050
) closing parenthesis	) closing parenthesis	52	52	0051
$ dollar sign	$ dollar sign	53	53	0044
= equals	= equals	54	54	0075
space	space	55	55	0040
, comma	, comma	56	56	0054
. period	. period	57	57	0056
# number sign	# number sign	60	60	0043
[opening bracket	[opening bracket	61	61	0133
] closing bracket	] closing bracket	62	62	0135
% percent sign††	% percent sign††	63††	63††	0045
" quotation mark	" quotation mark	64	64	0042
_ underline	_ underline	65	65	0137
¬ exclamation point	¬ exclamation point	66	66	0041
& ampersand	& ampersand	67	67	0046
' apostrophe	' apostrophe	70	70	0047
? question mark	? question mark	71	71	0077

TABLE A-1. INTERACTIVE TERMINAL CHARACTER SETS (Contd)

Character Sets		Code Sets		
ASCII Graphic (64-Character Set)	ASCII Character (128-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code
< less than	< less than	72	72	0074
> greater than	> greater than	73	73	0076
@ commmercial at		74		
\ reverse slant	\ reverse slant	75	75	0134
^ circumflex		76		
; semicolon	; semicolon	77	77	0073
	@ commercial at		7401	0100
	^ circumflex		7402	0136
	: colon††		7404††	0072
	` grave accent		7407	0140
	a		7601	0141
	b		7602	0142
	c		7603	0143
	d		7604	0144
	e		7605	0145
	f		7606	0146
	g		7607	1047
	h		7610	0150
	i		7611	0151
	j		7612	0152
	k		7613	0153
	l		7614	0154
	m		7615	0155
	n		7616	0156
	o		7617	0157
	p		7620	0160
	q		7621	0161
	r		7622	0162
	s		7623	0163
	t		7624	0164
	u		7625	0165
	v		7626	0166
	w		7627	0167
	x		7630	0170
	y		7631	0171
	z		7632	0172
	{ opening brace		7633	0173
	| vertical line		7634	0174
	} closing brace		7635	0175
	~ tilde		7636	0176
	NUL		7640	4000
	SOH		7641	0001
	STX		7642	0002
	ETX		7643	0003
	EOT		7644	0004
	ENQ		7645	0005
	ACK		7646	0006
	BEL		7647	0007
	BS		7650	0010
	HT		7651	0011
	LF		7652	0012
	VT		7653	0013
	FF		7654	0014
	CR		7655	0015
	SO		7656	0016
	SI		7657	0017
	DEL		7637	0177
	DLE		7660	0020

TABLE A-1. INTERACTIVE TERMINAL CHARACTER SETS (Contd)

Character Sets		Code Sets		
ASCII Graphic (64-Character Set)	ASCII Character (128-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code
	DC1		7661	0021
	DC2		7662	0022
	DC3		7663	0023
	DC4		7664	0024
	NAK		7665	0025
	SYN		7666	0026
	ETB		7667	0027
	CAN		7670	0030
	EM		7671	0031
	SUB		7672	0032
	ESC		7673	0033
	FS		7674	0034
	GS		7675	0035
	RS		7676	0036
	US		7677	0037

†Available only on NOS or through BASIC on NOS/BE.

††Character or code interpretation depends on context. Refer to Character Set Anomalies in the text.

TABLE A-2. 7-BIT ASCII CODE AND CHARACTER SETS

128-Character Set

96-Character Subset

Graphic 64-Character Subset

b_7			0	0	0	0	1	1	1	1
b_6			0	0	1	1	0	0	1	1
b_5			0	1	0	1	0	1	0	1
Bits			0	1	2	3	4	5	6	7
b_4	b_3	b_2	b_1	Row ↓ / Column →							
0	0	0	0	0	NUL 000	DLE 020	SP 040	0 060	@ 100	P 120	` 140	p 160
0	0	0	1	1	SOH 001	DC1 021	! 041	1 061	A 101	Q 121	a 141	q 161
0	0	1	0	2	STX 002	DC2 022	" 042	2 062	B 102	R 122	b 142	r 162
0	0	1	1	3	ETX 003	DC3 023	# 043	3 063	C 103	S 123	c 143	s 163
0	1	0	0	4	EOT 004	DC4 024	$ 044	4 064	D 104	T 124	d 144	t 164
0	1	0	1	5	ENQ 005	NAK 025	% 045	5 065	E 105	U 125	e 145	u 165
0	1	1	0	6	ACK 006	SYN 026	& 046	6 066	F 106	V 126	f 146	v 166
0	1	1	1	7	BEL 007	ETB 027	' 047	7 067	G 107	W 127	g 147	w 167
1	0	0	0	8	BS 010	CAN 030	(050	8 070	H 110	X 130	h 150	x 170
1	0	0	1	9	HT 011	EM 031	) 051	9 071	I 111	Y 131	i 151	y 171
1	0	1	0	A	LF 012	SUB 032	* 052	: 072	J 112	Z 132	j 152	z 172
1	0	1	1	B	VT 013	ESC 033	+ 053	; 073	K 113	[133	k 153	{ 173
1	1	0	0	C	FF 014	FS 034	, 054	< 074	L 114	\ 134	l 154	| 174
1	1	0	1	D	CR 015	GS 035	- 055	= 075	M 115	] 135	m 155	} 175
1	1	1	0	E	SO 016	RS 036	. 056	> 076	N 116	^ 136	n 156	~ 176
1	1	1	1	F	SI 017	US 037	/ 057	? 077	O 117	_ 137	o 157	DEL† 177

†The graphic 95-character subset does not include DEL; refer to Terminal Transmission Modes in the text.

LEGEND:

Numbers under characters are the octal values for the 7-bit character codes used within the network.

TABLE A-3. LOCAL BATCH DEVICE CHARACTER SETS

Character Sets			Code Sets			Card Keypunch Code	
CDC Graphic (64-Character Set)	ASCII Graphic (64-Character Set)	ASCII Graphic (95-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code	026	029
: colon††	: colon††		00††			8-2	8-2
A	A	A	01	01	0101	12-1	12-1
B	B	B	02	02	0102	12-2	12-2
C	C	C	03	03	0103	12-3	12-3
D	D	D	04	04	0104	12-4	12-4
E	E	E	05	05	0105	12-5	12-5
F	F	F	06	06	0106	12-6	12-6
G	G	G	07	07	0107	12-7	12-7
H	H	H	10	10	0110	12-8	12-8
I	I	I	11	11	0111	12-9	12-9
J	J	J	12	12	0112	11-1	11-1
K	K	K	13	13	0113	11-2	11-2
L	L	L	14	14	0114	11-3	11-3
M	M	M	15	15	0115	11-4	11-4
N	N	N	16	16	0116	11-5	11-5
O	O	O	17	17	0117	11-6	11-6
P	P	P	20	20	0120	11-7	11-7
Q	Q	Q	21	21	0121	11-8	11-8
R	R	R	22	22	0122	11-9	11-9
S	S	S	23	23	0123	0-2	0-2
T	T	T	24	24	0124	0-3	0-3
U	U	U	25	25	0125	0-4	0-4
V	V	V	26	26	0126	0-5	0-5
W	W	W	27	27	0127	0-6	0-6
X	X	X	30	30	0130	0-7	0-7
Y	Y	Y	31	31	0131	0-8	0-8
Z	Z	Z	32	32	0132	0-9	0-9
0	0	0	33	33	0060	0	0
1	1	1	34	34	0061	1	1
2	2	2	35	35	0062	2	2
3	3	3	36	36	0063	3	3
4	4	4	37	37	0064	4	4
5	5	5	40	40	0065	5	5
6	6	6	41	41	0066	6	6
7	7	7	42	42	0067	7	7
8	8	8	43	43	0070	8	8
9	9	9	44	44	0071	9	9
+ plus	+ plus	+ plus	45	45	0053	12	12-8-6
- hyphen (minus)	- hyphen (minus)	- hyphen (minus)	46	46	0055	11	11
* asterisk	* asterisk	* asterisk	47	47	0052	11-8-4	11-8-4
/ slant	/ slant	/ slant	50	50	0057	0-1	0-1
(open. paren.	(open. paren.	(open. paren.	51	51	0050	0-8-4	12-8-5
) clos. paren.	) clos. paren.	) clos. paren.	52	52	0051	12-8-4	11-8-5
$ dollar sign	$ dollar sign	$ dollar sign	53	53	0044	11-8-3	11-8-3
= equals	= equals	= equals	54	54	0075	8-3	8-6
space	space	space	55	55	0040	no punch	no punch
, comma	, comma	, comma	56	56	0054	0-8-3	0-8-3
. period	. period	. period	57	57	0056	12-8-3	12-8-3
≡ equivalence	# number sign	# number sign	60	60	0043	0-8-6	8-3
[open. bracket	[open. bracket	[open. bracket	61	61	0133	8-7	12-8-2 or 12-0†††
] clos. bracket	] clos. bracket	] clos. bracket	62	62	0135	0-8-2	11-8-2 or 11-0†††
% percent sign††	% percent sign††	% percent sign††	63††	63††	0045	8-6	0-8-4

TABLE A-3. LOCAL BATCH DEVICE CHARACTER SETS (Contd)

Character Sets			Code Sets			Card Keypunch Code	
CDC Graphic (64-Character Set)	ASCII Graphic (64-Character Set)	ASCII Graphic (95-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code	026	029
= not equals	" quotation mark	" quotation mark	64	64	0042	8-4	8-7
↱ concatenation	_ underline	_ underline	65	65	0137	0-8-5	0-8-5
∨ logical OR	! exclamation pt.	! exclamation pt.	66	66	0041	11-0	12-8-7
						or 11-8-2§	or 11-0§
∧ logical AND	& ampersand	& ampersand	67	67	0046	0-8-7	12
↑ superscript	' apostrophe	' apostrophe	70	70	0047	11-8-5	8-5
↓ subscript	? question mark	? question mark	71	71	0077	11-8-6	0-8-7
< less than	< less than	< less than	72	72	0074	12-0	12-8-4
						or 12-8-2§	or 12-0§
> greater than	> greater than	> greater than	73	73	0076	11-8-7	0-8-6
≤ less/equal	@ commercial at		74			8-5	8-4
≥ greater/equal	\ reverse slant	\ reverse slant	75	75	0134	12-8-5	0-8-2
¬ logical NOT	∧ circumflex		76			12-8-6	11-8-7
; semicolon	; semicolon	; semicolon	77	77	0073	12-8-7	11-8-6
		@ commercial at		7401	0100		
		∧ circumflex		7402	0136		
		: colon††		7404††	0072		
		` grave accent		7407	0140		
		a		7601	0141		
		b		7602	0142		
		c		7603	0143		
		d		7604	0144		
		e		7605	0145		
		f		7606	0146		
		g		7607	0147		
		h		7610	0150		
		i		7611	0151		
		j		7612	0152		
		k		7613	0153		
		l		7614	0154		
		m		7615	0155		
		n		7616	0156		
		o		7617	0157		
		p		7620	0160		
		q		7621	0161		
		r		7622	0162		
		s		7623	0163		
		t		7624	0164		
		u		7625	0165		
		v		7626	0166		
		w		7627	0167		
		x		7630	0170		
		y		7631	0171		
		z		7632	0172		
		{ open. brace		7633	0173		
		| vertical line		7634	0174		
		} clos. brace		7635	0175		
		~ tilde		7636	0176		

†Available only on NOS or through BASIC on NOS/BE.

††Character or code interpretation depends on context. Refer to Character Set Anomalies in the text.

†††Available for input only, on NOS.

§Available for input only, on NOS/BE or SCOPE 2.

TABLE A-4. 7-TRACK CODED TAPE CONVERSIONS

External BCD	ASCII Character	Octal 6-Bit Display Code	External BCD	ASCII Character	Octal 6-Bit Display Code
01	1	34	40	- hyphen (minus)	46
02	2	35	41	J	12
03	3	36	42	K	13
04	4	37	43	L	14
05	5	40	44	M	15
06	6	41	45	N	16
07	7	42	46	O	17
10	8	43	47	P	20
11	9	44	50	Q	21
12†	0	33	51	R	22
13	= equals	54	52	! exclamation point	66
14	" quotation mark	64	53	$ dollar sign	53
15	@ commercial at	74	54	* asterisk	47
16†	% percent sign	63	55	' apostrophe	70
17	[opening bracket	61	56	? question mark	71
20	space	55	57	> greater than	73
21	/ slant	50	60	+ plus	45
22	S	23	61	A	01
23	T	24	62	B	02
24	U	25	63	C	03
25	V	26	64	D	04
26	W	27	65	E	05
27	X	30	66	F	06
30	Y	31	67	G	07
31	Z	32	70	H	10
32	] closing bracket	62	71	I	11
33	, comma	56	72	< less than	72
34	(opening parenthesis	51	73	. period	57
35	_ underline	65	74	) closing parenthesis	52
36	# number sign	60	75	\ reverse slant	75
37	& ampersand	67	76	^ caret	76
			77	; semicolon	77

†As the text explains, conversion of these codes depends on whether the tape is read or written.

TABLE A-5. ASCII 9-TRACK CODED TAPE CONVERSION

ASCII				6-Bit Display Code†††	
Code Conversion†		Character and Code Conversion††			
Code (Hex)	Character	Code (Hex)	Character	ASCII Character	Code (Octal)
20	space	00	NUL	space	55
21	! exclamation point	7D	} closing brace	! exclamation point	66
22	" quotation mark	02	STX	" quotation mark	64
23	# number sign	03	ETX	# number sign	60
24	$ dollar sign	04	EOT	$ dollar sign	53
25	% percent sign§	05	ENQ	% percent sign§	63§
26	& ampersand	06	ACK	& ampersand	67
27	' apostrophe	07	BEL	' apostrophe	70
28	(opening parenthesis	08	BS	(opening parenthesis	51
29	) closing parenthesis	09	HT	) closing parenthesis	52
2A	* asterisk	0A	LF	* asterisk	47
2B	+ plus	0B	VT	+ plus	45
2C	, comma	0C	FF	, comma	56
2D	- hyphen (minus)	0D	CR	- hyphen (minus)	46
2E	. period	0E	SO	. period	57
2F	/ slant	0F	SI	/ slant	50
30	0	10	DLE	0	33
31	1	11	DC1	1	34
32	2	12	DC2	2	35
33	3	13	DC3	3	36
34	4	14	DC4	4	37
35	5	15	NAK	5	40
36	6	16	SYN	6	41
37	7	17	ETB	7	42
38	8	18	CAN	8	43
39	9	19	EM	9	44
3A	: colon§	1A	SUB	: colon§	00§
3B	; semicolon	1B	ESC	; semicolon	77
3C	< less than	7B	{ opening brace	< less than	72
3D	= equals	1D	GS	= equals	54
3E	> greater than	1E	RS	> greater than	73
3F	? question mark	1F	US	? question mark	71
40	@ commercial at	60	` grave accent	@ commercial at	74
41	A	61	a	A	01
42	B	62	b	B	02
43	C	63	c	C	03
44	D	64	d	D	04
45	E	65	e	E	05
46	F	66	f	F	06

TABLE A-5. ASCII 9-TRACK CODED TAPE CONVERSION (Contd)

ASCII				6-Bit Display Code†††	
Code Conversion†		Character and Code Conversion††			
Code (Hex)	Character	Code (Hex)	Character	ASCII Character	Code (Octal)
47	G	67	g	G	07
48	H	68	h	H	10
49	I	69	i	I	11
4A	J	6A	j	J	12
4B	K	6B	k	K	13
4C	L	6C	l	L	14
4D	M	6D	m	M	15
4E	N	6E	n	N	16
4F	O	6F	o	O	17
50	P	70	p	P	20
51	Q	71	q	Q	21
52	R	72	r	R	22
53	S	73	s	S	23
54	T	74	t	T	24
55	U	75	u	U	25
56	V	76	v	V	26
57	W	77	w	W	27
58	X	78	x	X	30
59	Y	79	y	Y	31
5A	Z	7A	z	Z	32
5B	[opening bracket	1C	FS	[opening bracket	61
5C	\ reverse slant	7C	| vertical line	\ reverse slant	75
5D	] closing bracket	01	SOH	] closing bracket	62
5E	^ caret	7E	~ tilde	^ caret	76
5F	_ underline	7F	DEL	_ underline	65

†When these characters are copied from or to a tape, the characters remain the same and the code changes from or to ASCII to or from display code.

††These characters do not exist in display code. When the characters are copied from a tape, each ASCII character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 61 (hexadecimal), from tape, it writes an uppercase A, 01 (octal).

†††A display code space always translates to an ASCII space.

§Character or code interpretation depends on context. Refer to Character Set Anomalies in the text.

TABLE A-6. EBCDIC 9-TRACK CODED TAPE CONVERSION

EBCDIC				6-Bit Display Code†††	
Code Conversion†		Character and Code Conversion††			
Code (Hex)	Character	Code (Hex)	Character	ASCII Character	Code (Octal)
40	space	00	NUL	space	55
4A	¢ cent sign	1C	IFS	[opening bracket	61
4B	. period	0E	SO	. period	57
4C	< less than	C0	{ opening brace	< less than	72
4D	(opening parenthesis	16	BS	(opening parenthesis	51
4E	+ plus	0B	VT	+ plus	45
4F	| vertical line	D0	} closing brace	! exclamation point	66
50	& ampersand	2E	ACK	& ampersand	67
5A	! exclamation point	01	SOH	] closing bracket	62
5B	$ dollar sign	37	EOT	$ dollar sign	53
5C	* asterisk	25	LF	* asterisk	47
5D	) closing parenthesis	05	HT	) closing parenthesis	52
5E	; semicolon	27	ESC	; semicolon	77
5F	¬ logical NOT	A1	~ tilde	^ caret	76
60	- hyphen (minus)	0D	CR	- hyphen (minus)	46
61	/ slant	0F	SI	/ slant	50
6B	, comma	0C	FF	, comma	56
6C	% percent sign§	2D	ENQ	% percent sign§	63§
6D	_ underline	07	DEL	_ underline	65
6E	> greater than	1E	IRS	> greater than	73
6F	? question mark	1F	IUS	? question mark	71
7A	: colon§	3F	SUB	: colon§	00§
7B	# number sign	03	ETX	# number sign	60
7C	@ commercial at	79	\ reverse slant	@ commercial at	74
7D	' apostrophe	2F	BEL	' apostrophe	70
7E	= equals	1D	IGS	= equals	54
7F	" quotation mark	02	STX	" quotation mark	64
C1	A	81	a	A	01
C2	B	82	b	B	02
C3	C	83	c	C	03
C4	D	84	d	D	04
C5	E	85	e	E	05
C6	F	86	f	F	06
C7	G	87	g	G	07
C8	H	88	h	H	10
C9	I	89	i	I	11
D1	J	91	j	J	12
D2	K	92	k	K	13
D3	L	93	l	L	14

TABLE A-6. EBCDIC 9-TRACK CODED TAPE CONVERSION (Contd)

EBCDIC				6-Bit Display Code†††	
Code Conversion†		Character and Code Conversion††			
Code (Hex)	Character	Code (Hex)	Character	ASCII Character	Code (Octal)
D4	M	94	m	M	15
D5	N	95	n	N	16
D6	O	96	o	O	17
D7	P	97	p	P	20
D8	Q	98	q	Q	21
D9	R	99	r	R	22
E0	\ reverse slant	6A	| vertical line	\ reverse slant	75
E2	S	A2	s	S	23
E3	T	A3	t	T	24
E4	U	A4	u	U	25
E5	V	A5	v	V	26
E6	W	A6	w	W	27
E7	X	A7	x	X	30
E8	Y	A8	y	Y	31
E9	Z	A9	z	Z	32
F0	0	10	DLE	0	33
F1	1	11	DC1	1	34
F2	2	12	DC2	2	35
F3	3	13	TM	3	36
F4	4	3C	DC4	4	37
F5	5	3D	NAK	5	40
F6	6	32	SYN	6	41
F7	7	26	ETB	7	42
F8	8	18	CAN	8	43
F9	9	19	EM	9	44

†When these characters are copied from or to a tape, the characters remain the same (except EBCDIC codes 4A (hexadecimal), 4F (hexadecimal), 5A (hexadecimal), and 5F (hexadecimal)) and the code changes from or to EBCDIC to or from display code.

††These characters do not exist in display code. When the characters are copied from a tape, each EBCDIC character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 81 (hexadecimal), from tape, it writes an uppercase A, 01 (octal).

†††A display code space always translates to an EBCDIC space.

§Character or code interpretation depends on context. Refer to Character Set Anomalies in the text.

TABLE A-7. FULL EBCDIC CHARACTER SET

Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†
00	0000	NUL	4A	0112	¢ cent sign	A7	0247	x
01	0001	SOH	4B	0113	. period	A8	0250	y
02	0002	STX	4C	0114	< less than	A9	0251	z
03	0003	ETX	4D	0115	(open. paren.	AA	0252	undefined
04	0004	PF	4E	0116	+ plus	thru	thru	
05	0005	HT	4F	0117	| logical OR	BF	0277	undefined
06	0006	LC	50	0120	& ampersand	C0	0300	{ open. brace
07	0007	DEL	51	0121	undefined	C1	0301	A
08	0010	undefined	thru	thru		C2	0302	B
09	0011	undefined	59	0131	undefined	C3	0303	C
0A	0012	SMM	5A	0132	! exclam. point	C4	0304	D
0B	0013	VT	5B	0133	$ dollar sign	C5	0305	E
0C	0014	FF	5C	0134	* asterisk	C6	0306	F
0D	0015	CR	5D	0135	) clos. paren.	C7	0307	G
0E	0016	SO	5E	0136	; semicolon	C8	0310	H
0F	0017	SI	5F	0137	¬ logical NOT	C9	0311	I
10	0020	DLE	60	0140	- minus	CA	0312	undefined
11	0021	DC1	61	0141	/ slant	CB	0313	undefined
12	0022	DC2	62	0142	undefined	CC	0314	♪
13	0023	TM	thru	thru		CD	0315	undefined
14	0024	RES	69	0151	undefined	CE	0316	Ψ
15	0025	NL	6A	0152	¦ vertical line	CF	0317	undefined
16	0026	BS	6B	0153	, comma	D0	0320	} clos. brace
17	0027	IL	6C	0154	% percent sign	D1	0321	J
18	0030	CAN	6D	0155	_ underline	D2	0322	K
19	0031	EM	6E	0156	> greater than	D3	0323	L
1A	0032	CC	6F	0157	? question mark	D4	0324	M
1B	0033	CU1	70	0160	undefined	D5	0325	N
1C	0034	IFS	thru	thru		D6	0326	O
1D	0035	IGS	78	0170	undefined	D7	0327	P
1E	0036	IRS	79	0171	` grave accent	D8	0330	Q
1F	0037	IUS	7A	0172	: colon	D9	0331	R
20	0040	DS	7B	0173	# number sign	DA	0332	undefined
21	0041	SOS	7C	0174	@ commercial at	thru	thru	
22	0042	FS	7D	0175	' apostrophe	DF	0337	undefined
23	0043	undefined	7E	0176	= equals	E0	0340	\ reverse slant
24	0044	BYP	7F	0177	" quotation mark	E1	0341	undefined
25	0045	LF	80	0200	undefined	E2	0342	S
26	0046	ETBB	81	0201	a	E3	0343	T
27	0047	ESCE	82	0202	b	E4	0344	U

TABLE A-7. FULL EBCDIC CHARACTER SET (Contd)

Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic or Control Character†
28	0050	undefined	83	0203	c	E5	0345	V
29	0051	undefined	84	0204	d	E6	0346	W
2A	0052	SM	85	0205	e	E7	0347	X
2B	0053	CU2	86	0206	f	E8	0350	Y
2C	0054	undefined	87	0207	g	E9	0351	Z
2D	0055	ENQ	88	0210	h	EA	0352	undefined
2E	0056	ACK	89	0211	i	EB	0353	undefined
2F	0057	BEL	8A	0212	undefined	EC	0354	undefined
30	0060	undefined	thru	thru		ED	0355	undefined
31	0061	undefined	90	0220	undefined	thru	thru	
32	0062	SYN	91	0221	j	EF	0357	undefined
33	0063	undefined	92	0222	k	F0	0360	0
34	0064	PN	93	0223	l	F1	0361	1
35	0065	RS	94	0224	m	F2	0362	2
36	0066	UC	95	0225	n	F3	0363	3
37	0067	EOT	96	0226	o	F4	0364	4
38	0070	undefined	97	0227	p	F5	0365	5
39	0071	undefined	98	0230	q	F6	0366	6
3A	0072	undefined	99	0231	r	F7	0367	7
3B	0073	CU3	9A	0232	undefined	F8	0370	8
3C	0074	DC4	thru	thru		F9	0372	9
3D	0075	NAK	A0	0240	undefined	FA	0372	| vertical line
3E	0076	undefined	A1	0241	~ tilde	FB	0373	undefined
3F	0077	SUB	A2	0242	s	thru	thru	
40	0100	space	A3	0243	t	FF	0377	undefined
41	0101	undefined	A4	0244	u			
thru	thru		A5	0245	v			
49	0111	undefined	A6	0246	w			

†Graphic characters shown are those used on the IBM System/370 standard (PN) print train. Other devices support subsets or variations of this character graphic set.

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 9, 14, 16, AND 17 (HASP, HPRE, 2780, AND 3780)

Terminal EBCDIC			Network ASCII (Normalized Mode)		
Octal Code	Graphic†	Control Character	Octal Code††	Graphic	Control Character
000		NUL	000		null
001		SOH	001		start of header
002		STX	002		start of text
003		ETX	003		end of text
004		PF	040	space	
005		HT	011		horizontal tabulate
006		LC	040	space	
007		DEL	177		delete
010		undefined	040	space	
011		undefined	040	space	
012		SMM	040	space	
013		VT	013		vertical tabulate
014		FF	014		form feed
015		CR	015		carriage return
016		SO	016		shift out
017		SI	017		shift in
020		DLE	020		data link escape
021		DC1	021		device control 1
022		DC2	022		device control 2
023		TM	023		device control 3
024		RES	040	space	
025		NL	040	space	
026		BS	010		backspace
027		IL	040	space	
030		CAN	030		cancel
031		EM	031		end of medium
032		CC	040	space	
033		CU1	040	space	
034		IFS	034		file separator
035		IGS	035		group separator
036		IRS	036		record separator
037		IUS	037		unit separator
040		DS	040	space	
041		SOS	040	space	
042		FS	040	space	
043		undefined	040	space	
044		BYP	040	space	
045		LF	012		linefeed
046		ETB or EOB	027		end of transmission block
047		ESC or PRE	033		escape
050		undefined	040	space	
051		undefined	040	space	
052		SM	040	space	
053		CU2	040	space	
054		undefined	040	space	
055		ENQ	005		enquiry
056		ACK	006		positive acknowledgment
057		BEL	007		bell
060		undefined	040	space	
061		undefined	040	space	
062		SYN	026		synchronous idle
063		undefined	040	space	
064		PN	040	space	
065		RS	040	space	
066		UC	040	space	
067		EOT	004		end of transmission
070		undefined	040	space	
071		undefined	040	space	
072		undefined	040	space	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 9, 14, 16, AND 17 (Contd)
(HASP, HPRE, 2780, AND 3780)

Terminal EBCDIC			Network ASCII (Normalized Mode)		
Octal Code	Graphic†	Control Character	Octal Code††	Graphic	Control Character
073		CU3	040	space	
074		DC4	024		device control 4
075		NAK	025		negative acknowledgment
076		undefined	040	space	
077		SUB	032		substitute
100	space		040	space	
101 thru 111		undefined	040	space	
112	¢		133	[	
113	.		056	.	
114	<		074	<	
115	(		050	(	
116	+		053	+	
117	|		041	!	
120	&		046	&	
121 thru 131		undefined	040	space	
132	!		135	]	
133	$		044	$	
134	*		052	*	
135	)		051	)	
136	;		073	;	
137	¬		136	^	
140	-		055	-	
141	/		057	/	
142 thru 151		undefined	040	space	
152	¦		174	¦	
153	,		054	,	
154	%		045	%	
155			137		
156	>		076	>	
157	?		077	?	
160 thru 170		undefined	040	space	
171	`		140	`	
172	:		172	:	
173	#		043	#	
174	@		100	@	
175	'		047	'	
176	=		075	=	
177	"		042	"	
200		undefined	040	space	
201	a		141	a	
202	b		142	b	
203	c		143	c	
204	d		144	d	
205	e		145	e	
206	f		146	f	
207	g		147	g	
210	h		150	h	
211	i		151	i	
212 thru 220		undefined	040	space	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 9, 14, 16, AND 17 (Contd)
(HASP, HPRE, 2780, AND 3780)

Terminal EBCDIC			Network ASCII (Normalized Mode)		
Octal Code	Graphic†	Control Character	Octal Code††	Graphic	Control Character
221	j		152	j	
222	k		153	k	
223	l		154	l	
224	m		155	m	
225	n		156	n	
226	o		157	o	
227	p		160	p	
230	q		161	q	
231	r		162	r	
232 thru 240		undefined	040	space	
241	~		176	~	
242	s		163	s	
243	t		164	t	
244	u		165	u	
245	v		166	v	
246	w		167	w	
247	x		170	x	
250	y		171	y	
251	z		172	z	
252 thru 277		undefined	040	space	
300	{		173	{	
301	A		101	A	
302	B		102	B	
303	C		103	C	
304	D		104	D	
305	E		105	E	
306	F		106	F	
307	G		107	G	
310	H		110	H	
311	I		111	I	
312		undefined	040	space	
313		undefined	040	space	
314	♪		040	space	
315		undefined	040	space	
316	↓		040	space	
317		undefined	040	space	
320	}		175	}	
321	J		112	J	
322	K		113	K	
323	L		114	L	
324	M		115	M	
325	N		116	N	
326	O		117	O	
327	P		120	P	
330	Q		121	Q	
331	R		122	R	
332 thru 337		undefined	040	space	
340	\		134	\	
341		undefined	040	space	
342	S		123	S	
343	T		124	T	
344	U		125	U	
345	V		126	V	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 9, 14, 16, AND 17 (Contd)
(HASP, HPRE, 2780, AND 3780)

Terminal EBCDIC			Network ASCII (Normalized Mode)		
Octal Code	Graphic†	Control Character	Octal Code††	Graphic	Control Character
346	W		127	W	
347	X		130	X	
350	Y		131	Y	
351	Z		132	Z	
352		undefined	040	space	
353		undefined	040	space	
354	┤		040	space	
355 thru 357		undefined	040	space	
360	0		060	0	
361	1		061	1	
362	2		062	2	
363	3		063	3	
364	4		064	4	
365	5		065	5	
366	6		066	6	
367	7		067	7	
370	8		070	8	
371	9		071	9	
372	|		040	space	
373 thru 377		undefined	040	space	

†Graphic characters shown are those used on the IBM System/370 standard (PN) print train. Other devices support subsets or variations of this character graphic set.

††Shown with zero parity (eighth or uppermost bit is always zero).

TABLE A-9. AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH 029 PUNCHED CARD CODES AND EBCDIC TRANSLATION (BATCH OUTPUT DEVICES, TERMINAL CLASSES 9, 14, 16, AND 17) (HASP, HPRE, 2780, AND 3780)

ASCII Bit b_4 b_3 b_2 b_1	Row \ Col b_8 b_7 b_6 b_5	2 (0 0 1 0)	3 (0 0 1 1)	4 (0 1 0 0)	5 (0 1 0 1)
0 0 0 0	0	SP no punch SP 40	0 0 0 F0	@ 8-4 @ 7C	P 11-7 P D7
0 0 0 1	1	! 12-8-7 ! 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8
0 0 1 0	2	" 8-7 " 7F	2 2 2 F2	B 12-2 B C2	R 11-9 R D9
0 0 1 1	3	# 8-3 # 7B	3 3 3 F3	C 12-3 C C3	S 0-2 S E2
0 1 0 0	4	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3
0 1 0 1	5	% 0-8-4 % 6C	5 5 5 F5	E 12-5 E C5	U 0-4 U E4
0 1 1 0	6	& 12 & 50	6 6 6 F6	F 12-6 F C6	V 0-5 V E5
0 1 1 1	7	' 8-5 ' 7D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6

ASCII Bit b_4 b_3 b_2 b_1	Row \ Col b_8 b_7 b_6 b_5	2 (0 0 1 0)	3 (0 0 1 1)	4 (0 1 0 0)	5 (0 1 0 1)
1 0 0 0	8	(12-8-5 (4D	8 8 8 F8	H 12-8 H C8	X 0-7 X E7
1 0 0 1	9	) 11-8-5) 5D	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8
1 0 1 0	10 (A)	* 11-8-4 * 5C	: 8-2 : 7A	J 11-1 J D1	Z 0-9 Z E9
1 0 1 1	11 (B)	+ 12-8-6 + 4E	; 11-8-6 ; 5E	K 11-2 K D2	[12-8-2 ¢ 4A
1 1 0 0	12 (C)	, 0-8-3 , 6B	< 12-8-4 < 4C	L 11-3 L D3	\ 0-8-2 \ E0
1 1 0 1	13 (D)	- 11 - 60	= 8-6 = 7E	M 11-4 M D4	] 11-8-2 ! 5A
1 1 1 0	14 (E)	. 12-8-3 . 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ¬ 5F
1 1 1 1	15 (F)	/ 0-1 / 61	? 0-8-7 ? 6F	O 11-6 O D6	_ 0-8-5 _ 6D

LEGEND:

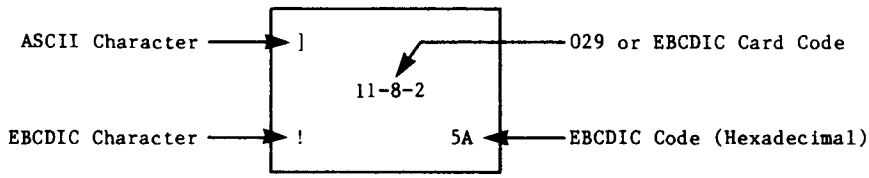
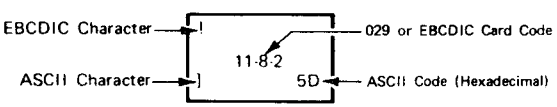


TABLE A-10. EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC) WITH 029 PUNCHED CARD CODES AND ASCII TRANSLATION (BATCH INPUT DEVICES, TERMINAL CLASSES 9, 14, 16, AND 17) (HASP, HPRE, 2780, AND 3780)

EBCDIC BITS 8 7 6 5	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
4 3 2 1 / 1ST HEX 2ND	0	1	2	3	4	5	6	7	8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
0 0 0 0 / 0	NUL 12-0-9-8-1 SP 20	DLE 12-11-9-8-1 SP 20	DS 11-0-9-8-1 SP 20	12-11-0-9-8-1 SP 20	SP no punch SP 20	& 12 & 26	- 11 - 2D	12-11-0 SP 20	12-0-8-1 SP 20	12-11-8-1 SP 20	11-0-8-1 SP 20	12-11-0-8-1 SP 20	{ 12-0 < 3C	} 11-0 ! 21	\ 0-8-2 \ 5C	0 0 0 30
0 0 0 1 / 1	SOH 12-9-1 SP 20	DC1 11-9-1 SP 20	SOS 0-9-1 SP 20	9-1 SP 20	12-0-9-1 SP 20	12-11-9-1 SP 20	/ 0-1 / 2F	12-11-0-9-1 SP 20	a 12-0-1 A 41	j 12-11-1 J 4A	~ 11-0-1 ^ 5E	12-11-0-1 SP 20	A 12-1 A 41	J 11-1 J 4A	11-0-9-1 9F	1 1 1 31
0 0 1 0 / 2	STX 12-9-2 SP 20	DC2 11-9-2 SP 20	FS 0-9-2 SP 20	SYN 9-2 SP 20	12-0-9-2 SP 20	12-11-9-2 SP 20	11-0-9-2 SP 20	12-11-0-9-2 SP 20	b 12-0-2 B 42	k 12-11-2 K 4B	s 11-0-2 S 53	12-11-0-2 SP 20	B 12-2 B 42	K 11-2 K 4B	S 0-2 S 53	2 2 2 32
0 0 1 1 / 3	ETX 12-9-3 SP 20	TM 11-9-3 SP 20	0-9-3 SP 20	9-3 SP 20	12-0-9-3 SP 20	12-11-9-3 SP 20	11-0-9-3 SP 20	12-11-0-9-3 SP 20	c 12-0-3 C 43	l 12-11-3 L 4C	t 11-0-3 T 54	12-11-0-3 SP 20	C 12-3 C 43	L 11-3 L 4C	T 0-3 T 54	3 3 3 33
0 1 0 0 / 4	PF 12-9-4 SP 20	RES 11-9-4 SP 20	BYP 0-9-4 SP 20	PN 9-4 SP 20	12-0-9-4 SP 20	12-11-9-4 SP 20	11-0-9-4 SP 20	12-11-0-9-4 SP 20	d 12-0-4 D 44	m 12-11-4 M 4D	u 11-0-4 U 55	12-11-0-4 SP 20	D 12-4 D 44	M 11-4 M 4D	U 0-4 U 55	4 4 4 34
0 1 0 1 / 5	HT 12-9-5 SP 20	NL 11-9-5 SP 20	LF 0-9-5 SP 20	RS 9-5 SP 20	12-0-9-5 SP 20	12-11-9-5 SP 20	11-0-9-5 SP 20	12-11-0-9-5 SP 20	e 12-0-5 E 45	n 12-11-5 N 4E	v 11-0-5 V 56	12-11-0-5 SP 20	E 12-5 E 45	N 11-5 N 4E	V 0-5 V 56	5 5 5 35
0 1 1 0 / 6	LC 12-9-6 SP 20	BS 11-9-6 SP 20	ETB 0-9-6 SP 20	UC 9-6 SP 20	12-0-9-6 SP 20	12-11-9-6 SP 20	11-0-9-6 SP 20	12-11-0-9-6 SP 20	f 12-0-6 F 46	o 12-11-6 O 4F	w 11-0-6 W 57	12-11-0-6 SP 20	F 12-6 F 46	O 11-6 O 4F	W 0-6 W 57	6 6 6 36
0 1 1 1 / 7	DEL 12-9-7 SP 20	IL 11-9-7 SP 20	ESC 0-9-7 SP 20	EOT 9-7 SP 20	12-0-9-7 SP 20	12-11-9-7 SP 20	11-0-9-7 SP 20	12-11-0-9-7 SP 20	g 12-0-7 G 47	p 12-11-7 P 50	x 11-0-7 X 58	12-11-0-7 SP 20	G 12-7 G 47	P 11-7 P 50	X 0-7 X 58	7 7 7 37
1 0 0 0 / 8	GE 12-9-8 SP 20	CAN 11-9-8 SP 20	0-9-8 SP 20	9-8 SP 20	12-0-9-8 SP 20	12-11-9-8 SP 20	11-0-9-8 SP 20	12-11-0-9-8 SP 20	h 12-0-8 H 48	q 12-11-8 Q 51	y 11-0-8 Y 59	12-11-0-8 SP 20	H 12-8 H 48	Q 11-8 Q 51	Y 0-8 Y 59	8 8 8 38
1 0 0 1 / 9	RLF 12-9-8-1 SP 20	EM 11-9-8-1 SP 20	0-9-8-1 SP 20	9-8-1 SP 20	12-8-1 SP 20	11-8-1 SP 20	0-8-1 SP 20	8-1 @ 40	i 12-0-9 I 49	r 12-11-9 R 52	z 11-0-9 Z 5A	12-11-0-9 SP 20	I 12-9 I 49	R 11-9 R 52	Z 0-9 Z 5A	9 9 9 39
1 0 1 0 / A (10)	SMM 12-9-8-2 SP 20	CC 11-9-8-2 SP 20	SM 0-9-8-2 SP 20	9-8-2 SP 20	¢ 12-8-2 [5B	! 11-8-2] 5D	; 12-11 \ 5C	: 8-2 : 3A	12-0-8-2 SP 20	12-11-8-2 SP 20	11-0-8-2 SP 20	12-11-0-8-2 SP 20	12-0-9-8-2 SP 20	12-11-9-8-2 SP 20	11-0-9-8-2 SP 20	¦(LVM) 12-11-0-9-8-2 SP 20
1 0 1 1 / B (11)	VT 12-9-8-3 SP 20	CU1 11-9-8-3 SP 20	CU2 0-9-8-3 SP 20	CU3 9-8-3 SP 20	. 12-8-3 . 2E	$ 11-8-3 $ 24	, 0-8-3 , 2C	# 8-3 # 23	12-0-8-3 SP 20	12-11-8-3 SP 20	11-0-8-3 SP 20	12-11-0-8-3 SP 20	12-0-9-8-3 SP 20	12-11-9-8-3 SP 20	11-0-9-8-3 SP 20	12-11-0-9-8-3 SP 20
1 1 0 0 / C (12)	FF 12-9-8-4 SP 20	IFS 11-9-8-4 SP 20	0-9-8-4 SP 20	DC4 9-8-4 SP 20	< 12-8-4 < 3C	* 11-8-4 * 2A	% 0-8-4 % 25	@ 8-4 @ 40	12-0-8-4 SP 20	12-11-8-4 SP 20	11-0-8-4 SP 20	12-11-0-8-4 SP 20	↵ 12-0-9-8-4 SP 20	12-11-9-8-4 SP 20	┌ 11-0-9-8-4 SP 20	12-11-0-9-8-4 SP 20
1 1 0 1 / D (13)	CR 12-9-8-5 SP 20	IGS 11-9-8-5 SP 20	ENQ 0-9-8-5 SP 20	NAK 9-8-5 SP 20	(12-8-5 (28	) 11-8-5) 29	¬ 0-8-5 ¬ 5F	' 8-5 ' 27	12-0-8-5 SP 20	12-11-8-5 SP 20	11-0-8-5 SP 20	12-11-0-8-5 SP 20	12-0-9-8-5 SP 20	12-11-9-8-5 SP 20	11-0-9-8-5 SP 20	12-11-0-9-8-5 SP 20
1 1 1 0 / E (14)	SO 12-9-8-6 SP 20	IRS 11-9-8-6 SP 20	ACK 0-9-8-6 SP 20	9-8-6 SP 20	+ 12-8-6 + 2B	; 11-8-6 ; 3B	> 0-8-6 > 3E	= 8-6 = 3D	12-0-8-6 SP 20	12-11-8-6 SP 20	11-0-8-6 SP 20	12-11-0-8-6 SP 20	↳ 12-0-9-8-6 SP 20	12-11-9-8-6 SP 20	11-0-9-8-6 SP 20	12-11-0-9-8-6 SP 20
1 1 1 1 / F (15)	SI 12-9-8-7 SP 20	IUS 11-9-8-7 SP 20	BEL 0-9-8-7 SP 20	SUB 9-8-7 SP 20	¦ 12-8-7 ! 21	¬ 11-8-7 ^ 5E	? 0-8-7 ? 3F	" 8-7 " 22	12-0-8-7 SP 20	12-11-8-7 SP 20	11-0-8-7 SP 20	12-11-0-8-7 SP 20	12-0-9-8-7 SP 20	12-11-9-8-7 SP 20	11-0-9-8-7 SP 20	EO 12-11-0-9-8-7 SP 20

LEGEND



NOTE: A card with the character SUB punched in column 1 functions as a /*EOR separator card.

TABLE A-11. AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH 026 PUNCHED CARD CODES AND EBCDIC TRANSLATION (BATCH OUTPUT DEVICES, TERMINAL CLASSES 9, 14, 16, AND 17) (HASP, HPRE, 2780, AND 3780)

ASCII Bit b_4 b_3 b_2 b_1	Col / Row	b_8 b_7 b_6 b_5 = 0 0 1 0	b_8 b_7 b_6 b_5 = 0 0 1 1	b_8 b_7 b_6 b_5 = 0 1 0 0	b_8 b_7 b_6 b_5 = 0 1 0 1
		2	3	4	5
0 0 0 0	0	SP no punch SP 40	0 0 0 F0	<u><</u> 8-5 ' 7D	P 11-7 P D7
0 0 0 1	1	! 12-8-7 ! 4F	1 1 1 F1	A 12-1 A C1	Q 11-8 Q D8
0 0 1 0	2	≠ 8-4 @ 7C	2 2 2 F2	B 12-2 B C2	R 11-9 R D9
0 0 1 1	3	≡ 0-8-6 > 6E	3 3 3 F3	C 12-3 C C3	S 0-2 S E2
0 1 0 0	4	$ 11-8-3 $ 5B	4 4 4 F4	D 12-4 D C4	T 0-3 T E3
0 1 0 1	5	% 8-6 = 7E	5 5 5 F5	E 12-5 E C5	U 0-4 U E4
0 1 1 0	6	∧ 0-8-7 ? 6F	6 6 6 F6	F 12-6 F C6	V 0-5 V E5
0 1 1 1	7	↑ 0-8-5) 5D	7 7 7 F7	G 12-7 G C7	W 0-6 W E6

ASCII Bit b_4 b_3 b_2 b_1	Col / Row	b_8 b_7 b_6 b_5 = 0 0 1 0	b_8 b_7 b_6 b_5 = 0 0 1 1	b_8 b_7 b_6 b_5 = 0 1 0 0	b_8 b_7 b_6 b_5 = 0 1 0 1
		2	3	4	5
1 0 0 0	8	(0-8-4 % 6C	8 8 8 F8	H 12-8 H C8	X 0-7 X E7
1 0 0 1	9	) 12-8-4 < 4C	9 9 9 F9	I 12-9 I C9	Y 0-8 Y E8
1 0 1 0	10 (A)	* 11-8-4 * 5C	: 8-2 : 7A	J 11-1 J D1	Z 0-9 Z E9
1 0 1 1	11 (B)	+ 12 & 50	; 12-8-7 ↓ 4F	K 11-2 K D2	[8-7 " 7F
1 1 0 0	12 (C)	, 0-8-3 , 6B	< 12-0 { C0	L 11-3 L D3	<u>></u> 12-8-5 (4D
1 1 0 1	13 (D)	- 11 - 60	= 8-3 # 7B	M 11-4 M D4	] 0-8-2 \ E0
1 1 1 0	14 (E)	. 12-8-3 . 4B	> 11-8-7 ¬ 5F	N 11-5 N D5	¬ 12-8-6 + 4E
1 1 1 1	15 (F)	/ 0-1 / 61	↓ 11-8-6 ; 5E	O 11-6 O D6	→ 0-8-5 _ 6D

LEGEND:

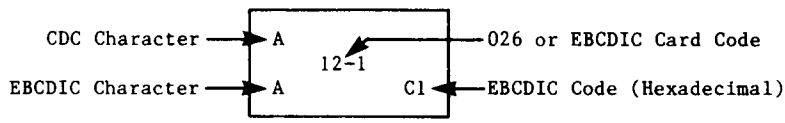


TABLE A-12. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, 751, H2000, T4014, AND M40)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
000		NUL or (@)	000		null
003	▲	ETX or (C)	003		end of text
005		ENQ or WRU or (E)	005		enquiry
006		ACK or RU or (F)	006		positive acknowledgment
011		HT or (I)	011		horizontal tabulate
012		LF or NL or ↓ or (J)	012		linefeed
014		FF or FORM or (L)	014		formfeed
017	⩾	SI or (O)	017		shift in
021		DC1 or X-ON or (Q)	021		device control 1
022		DC2 or TAPE or (R)	022		device control 2
024		DC4 or TAPE or (T)	024		device control 4
027		ETB or (W)	027		end transmission block
030		CAN or CLEAR or (X)	030		cancel
033		ESC or ESCAPE or ([)	033		escape
035		GS or (])	035		group separator
036		RS or (^)	036		record separator
041	!		041	!	
042	"		042	"	
044	$		044	$	
047	'		047	'	
050	(		050	(	
053	+		053	+	
055	-		055	-	
056	.		056	.	
060	0		060	0	
063	3		063	3	
065	5		065	5	
066	6		066	6	
071	9		071	9	
072	:		072	:	
074	<		074	<	
077	?		077	?	
101	A		101	A	
102	B		102	B	
104	D		104	D	
107	G		107	G	
110	H		110	H	
113	K		113	K	
115	M		115	M	
116	N		116	N	
120	P		120	P	
123	S		123	S	
125	U		125	U	
126	V		126	V	
131	Y		131	Y	
132	Z		132	Z	
134	\		134	\	
137	_ or ←		137		
140	`		140	`	
143	c		143	c	
145	e		145	e	
146	f		146	f	
151	i		151	i	
152	j		152	j	
154	l		154	l	
157	o		157	o	
161	q		161	q	
162	r		162	r	

TABLE A-12. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (Contd)
(M33, 713, 751, H2000, T4014, AND M40)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
164	t		164	t	
167	w		167	w	
170	x		170	x	
173	{		173	{	
174	¦ or ↑ or |		174	¦	
175	}		175	}	
176	~ or ¬		176	~	
201		SOH or (A)	001		start of header
202		STX or (B)	002		start of text
204		EOT or (D)	004		end of transmission
207		BELL or (G)	007		bell
210		BS or ← or (H)	010		backspace
213		VT or (K)	013		vertical tabulate
215		CR or RETURN or (M)	015		carriage return
216	≪	SO or (N)	016		shift out
220		DLE or (P)	020		data link escape
223		DC3 or X-OFF or (S)	023		device control 3
225		NAK or → or (U)	025		negative acknowledgment
226		SYN or LINE CLEAR or (V)	026		synchronous idle
231		EM or RESET or (Y)	031		end of medium
232		SUB or ↑ or (Z)	032		substitute
234		FS or (\)	034		file separator
237		US or (−)	037		unit separator
240	SPACE or blank		040	space	
243	#		043	#	
245	%		045	%	
246	&		046	&	
251	)		051	)	
252	*		052	*	
254	,		054	,	
257	/		057	/	
261	1		061	1	
262	2		062	2	
264	4		064	4	
267	7		067	7	
270	8		070	8	
273	;		073	;	
275	=		075	=	
276	>		076	>	
300	@		100	@	
303	C		103	C	
305	E		105	E	
306	F		106	F	
311	I		111	I	
312	J		112	J	
314	L		114	L	
317	O		117	O	
321	Q		121	Q	
322	R		122	R	
324	T		124	T	
327	W		127	W	
330	X		130	X	
333	[		133	[	
335	]		135	]	
336	^ or ¬		136	^	
341	a		141	a	

TABLE A-12. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (Contd)
(M33, 713, 751, H2000, T4014, AND M40)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
342	b		142	b	
344	d		144	d	
347	g		147	g	
350	h		150	h	
353	k		153	k	
355	m		155	m	
356	n		156	n	
360	p		160	p	
363	s		163	s	
365	u		165	u	
366	v		166	v	
371	y		171	y	
372	z		172	z	
377	■	DEL or RUBOUT	177		delete

†Shown with even parity, which is the default for these terminal classes (unless PA=N, an application program receives the same code as in normalized mode).

††A circle around a character indicates that the character key is pressed in conjunction with a CTL, CTRL, CNTRL, or CONTROL key to generate the code.

†††Shown with zero parity (eighth or uppermost bit is always zero).

TABLE A-13. CHARACTER CODE TRANSLATIONS, ASCII TERMINAL CLASSES 10 AND 15 (200UT AND 734)

Terminal ASCII†					Network ASCII (Normalized Mode)		
Octal Code††	Keyboard or Printer Graphic		029 Card Code	026 Card Code	Octal Code†††		Graphic
	ASCII	CDC			Input or Output	Console Output Only	
040	blank	blank	no punch	no punch	040		space
043	#		8-3	0-8-6	043		#
045	%	%	0-8-4	8-6	045		%
046	&		12	0-8-7	046		&
051	)	)	11-8-5	12-8-4	051		)
052	*	*	11-8-4	11-8-4	052		*
054	,	,	0-8-3	0-8-3	054		,
057	/	/	0-1	0-1	057		/
061	1	1	1	1	061		1
062	2	2	2	2	062		2
064	4	4	4	4	064		4
067	7	7	7	7	067		7
070	8	8	8	8	070		8
073	;	;	11-8-6	12-8-7	073		;
075	=	=	8-6	8-3	075		=
076	>	>	0-8-6	11-8-7	076		>
100	@	≤	8-4	11-8-5	100	140	@
103	C	C	12-3	12-3	103	143	C
105	E	E	12-5	12-5	105	145	E
106	F	F	12-6	12-6	106	146	F
111	I	I	12-9	12-9	111	151	I
112	J	J	11-1	11-1	112	152	J
114	L	L	11-3	11-3	114	154	L
117	O	O	11-6	11-6	117	157	O
121	Q	Q	11-8	11-8	121	161	Q
122	R	R	11-9	11-9	122	162	R
124	T	T	0-3	0-3	124	164	T
127	W	W	0-6	0-6	127	167	W
130	X	X	0-7	0-7	130	170	X
133	[	[	12-0 or 12-8-2	12-0 or 12-8-3	133	173	[
135	]	]	11-0 or 11-8-2	11-0 or 11-8-2	135	175	]
136	^	¬	11-8-7	12-8-6	136	176	^
241	!		12-8-7	0-8-2	041		!
242	"	≠	8-7	8-4	042		"
244	$	$	11-8-3	11-8-3	044		$
247	'		8-5	8-7	047		'
250	(	(	12-8-5	0-8-4	050		(

TABLE A-13. CHARACTER CODE TRANSLATIONS, ASCII TERMINAL CLASSES 10 AND 15 (Contd)
(200UT AND 734)

Terminal ASCII†					Network ASCII (Normalized Mode)		
Octal Code††	Keyboard or Printer Graphic		029 Card Code	026 Card Code	Octal Code†††		Graphic
	ASCII	CDC			Input or Output	Console Output Only	
253	+	+	12-8-6	12	053		+
255	-	-	11	11	055		-
256	.	.	12-8-3	12-8-3	056		.
260	0	0	0	0	060		0
263	3	3	3	3	063		3
265	5	5	5	5	065		5
266	6	6	6	6	066		6
271	9	9	9	9	071		9
272	:	:	8-2	8-2	072		:
274	<	<	12-8-4	8-5	074		<
277	?	↓	0-8-7	11-8-6	077		?
301	A	A	12-1	12-1	101	141	A
302	B	B	12-2	12-2	102	142	B
304	D	D	12-4	12-4	104	144	D
307	G	G	12-7	12-7	107	147	G
310	H	H	12-8	12-8	110	150	H
313	K	K	11-2	11-2	113	153	K
315	M	M	11-4	11-4	115	155	M
316	N	N	11-5	11-5	116	156	N
320	P	P	11-7	11-7	120	160	P
323	S	S	0-2	0-2	123	163	S
325	U	U	0-4	0-4	125	165	U
326	V	V	0-5	0-5	126	166	V
331	Y	Y	0-8	0-8	131	171	Y
332	Z	Z	0-9	0-9	132	172	Z
334	\	≥	0-8-2	12-8-5	134	174	\
337	_	↱	0-8-5	0-8-5	135	177	_

†Escape codes and control codes are not listed. These are not treated as network data and have no equivalent normalized mode translations.

††Shown with odd parity, the only possible parity selection for these terminal classes.

†††Shown with zero parity (eighth or uppermost bit is always zero). During output, codes 000 through 037_8 are converted to code 040_8 (blank). Codes for lowercase ASCII characters sent to the console are converted to the codes for the equivalent uppercase characters supported by the terminal, as shown; codes for these lowercase characters cannot be sent to batch devices without causing errors.

TABLE A-14. CHARACTER CODE TRANSLATIONS, BCD TERMINAL CLASSES 10 AND 15 (200UT AND 734)

Terminal External BCD†					Network ASCII (Normalized Mode)		
Octal Code††	Keyboard or Printer Graphic		029 Card Code	026 Card Code	Octal Code†††		Graphic
	ASCII	CDC			Input or Output	Console Output Only	
020	:	:	8-2	8-2	072		:
040	-	-	11	11	055		-
043	L	L	11-3	11-3	114	154	L
045	N	N	11-5	11-5	116	156	N
046	O	O	11-6	11-6	117	157	O
051	R	R	11-9	11-9	122	162	R
052	!	∨	12-8-7	11-0	041		!
054	*	*	11-8-4	11-8-4	052		*
057	>	>	0-8-6	11-8-7	076		>
061	A	A	12-1	12-1	101	141	A
062	B	B	12-2	12-2	102	142	B
064	D	D	12-4	12-4	104	144	D
067	G	G	12-7	12-7	107	147	G
070	H	H	12-8	12-8	108	148	H
073	.	.	12-8-3	12-8-3	056		.
075			12-0	12-8-5	134	174	\
103	3	3	3	3	063		3
105	5	5	5	5	065		5
106	6	6	6	6	066		6
111	9	9	9	9	071		9
112	0	0	0	0	060		0
114	"	=	8-5 or 8-7	8-4	042		"
117	[	[	8-4	8-7	133	173	[
121	/	/	0-1	0-1	057		/
122	S	S	0-2	0-2	123	163	S
124	U	U	0-4	0-4	125	165	U
127	X	X	0-7	0-7	130	170	X
130	Y	Y	0-8	0-8	131	171	Y
133	,	,	0-8-3	0-8-3	054		,
135	_	↱	0-8-5	0-8-5	137	177	_
136	#	≡	8-3	0-8-6	043		#
241	J	J	11-1	11-1	112	152	J
242	K	K	11-2	11-2	113	153	K
244	M	M	11-4	11-4	115	155	M
247	P	P	11-7	11-7	120	160	P
250	Q	Q	11-8	11-8	121	161	Q
253	$	$	11-8-3	11-8-3	044		$

TABLE A-14. CHARACTER CODE TRANSLATIONS, BCD TERMINAL CLASSES 10 AND 15 (Contd)
(200UT AND 734)

Terminal External BCD†					Network ASCII (Normalized Mode)		
Octal Code††	Keyboard or Printer Graphic		029 Card Code	026 Card Code	Octal Code†††		Graphic
	ASCII	CDC			Input or Output	Console Output Only	
255	'	↑	12	11-8-5	047		'
256	?	↓	12-8-7	11-8-6	077		?
263	C	C	12-3	12-3	103	143	C
265	E	E	12-5	12-5	105	145	E
266	F	F	12-6	12-6	106	146	F
271	I	I	12-9	12-9	111	151	I
272	<	<	12-8-4	12-0	074		<
274	)	)	11-8-5	12-8-4	051		)
277	;	;	11-8-6	12-8-7	073		;
301	1	1	1	1	061		1
302	2	2	2	2	062		2
304	4	4	4	4	064		4
307	7	7	7	7	067		7
310	8	8	8	8	070		8
313	=	=	8-6	8-3	075		=
315	@	≤	11-8-7	8-5	100	140	@
316	%	%	0-8-4	8-6	045		%
320	blank	blank	no punch	no punch	040		space
323	T	T	0-3	0-3	124	164	T
325	V	V	0-5	0-5	126	166	V
326	W	W	0-6	0-6	127	167	W
331	Z	Z	0-9	0-9	132	172	Z
332	]	]	0-8-2	0-8-2	135	175	]
334	(	(	12-8-5	0-8-4	050		(
337	&	∧	0-8-7	0-8-7	046		&
320	∧ or blank	¬ or ■ or none	none	none		136, 176	∧§

†Escape codes and control codes are not listed. These are not treated as network data and have no equivalent normalized mode translations.

††Shown with odd parity, the only possible parity selection for these terminal classes.

†††Shown with zero parity (eighth or uppermost bit is always zero). During output, codes 000 through 037_8 are converted to code 320_8 (blank). Codes for lowercase ASCII characters sent to the console are converted to the codes for the equivalent uppercase characters supported by the terminal, as shown; codes for these lowercase characters cannot be sent to batch devices without causing errors.

§Input and output of this symbol is not possible on some terminals. BCD transmission conventions support the rubout symbol ■ as an internal terminal memory parity error indicator instead. The ASCII codes 136_8 and 176_8 are output as a blank.

TABLE A-15. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 11, 12, AND 13 (711, 714, AND 714X)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
001		SOH or (A)	001		start of header
002		STX or (B)	002		start of text
004		EOT or (D)	004		end of transmission
007		BELL or (G)	007		bell
010		BS or ← or (H)	010		backspace
013		VT or (K)	013		vertical tabulate
015		CR or RETURN or (M)	015		carriage return
016		SO or (N)	016		shift out
020		DLE or (P)	020		data link escape
025		NAK or → or (U)	025		negative acknowledgment
026		SYN or LINE CLEAR or (V)	026		synchronous idle
031		EM or RESET or (Y)	031		end of medium
032		SUB or ↑ or (Z)	032		substitute
034		FS or (\)	034		file separator
037		US or (-)	037		unit separator
040	SPACE or blank		040	space	
043	#		043	#	
045	%		045	%	
046	&		046	&	
051	)		051	)	
052	*		052	*	
054	,		054	,	
057	/		057	/	
061	1		061	1	
062	2		062	2	
064	4		064	4	
067	7		067	7	
070	8		070	8	
073	;		073	;	
075	=		075	=	
076	>		076	>	
100	@		100	@	
103	C		103	C	
105	E		105	E	
106	F		106	F	
111	I		111	I	
112	J		112	J	
114	L		114	L	
117	O		117	O	
121	Q		121	Q	
122	R		122	R	
124	T		124	T	
127	W		127	W	
130	X		130	X	
133	[		133	[	
135	]		135	]	
136	∧ or ¬		136	∧	
141	a		141	a	
142	b		142	b	
144	d		144	d	
147	g		147	g	
150	h		150	h	
153	k		153	k	
155	m		155	m	
156	n		156	n	
160	p		160	p	

TABLE A-15. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 11, 12, AND 13 (Contd)
(711, 714, AND 714X)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
163	s		163	s	
165	u		165	u	
166	v		166	v	
171	y		171	y	
172	z		172	z	
177	■	DEL or RUBOUT	177		delete
200		NUL or (@)	000		null
203		ETX or (C) or SEND	003		end of text
205		ENQ or WRU or (E)	005		enquiry
206		ACK or RU or (F)	006		positive acknowledgment
211		HT or (I)	011		horizontal tabulate
212		LF or NL or ↓ or (J) or NEW LINE	012		linefeed
214		FF or FORM or (L)	014		formfeed
217		SI or (O)	017		shift in
221		DC1 or X-ON or (Q)	021		device control 1
222		DC2 or TAPE or (R)	022		device control 2
223		DC3 or X-OFF or (S)	023		device control 3
224		DC4 or TAPE or (T)	024		device control 4
227		ETB or (W)	027		end transmission block
230		CAN or CLEAR or (X)	030		cancel
233		ESC or ESCAPE or ([)	033		escape
235		GS or (])	035		group separator
236		RS or (^)	036		record separator
241	!		041	!	
242	"		042	"	
244	$		044	$	
247	'		047	'	
250	(		050	(	
253	+		053	+	
255	-		055	-	
256	.		056	.	
260	0		060	0	
263	3		063	3	
265	5		065	5	
266	6		066	6	
271	9		071	9	
272	:		072	:	
274	<		074	<	
277	?		077	?	
301	A		101	A	
302	B		102	B	
304	D		104	D	
307	G		107	G	
310	H		110	H	
313	K		113	K	
315	M		115	M	
316	N		116	N	
320	P		120	P	
323	S		123	S	
325	U		125	U	
326	V		126	V	
331	Y		131	Y	
332	Z		132	Z	
334	\		134	\	
337	_ or ←		137		
340	~		140	~	
343	c		143	c	

TABLE A-15. CHARACTER CODE TRANSLATIONS, CONSOLES IN TERMINAL CLASSES 11, 12, AND 13 (Contd)
(711, 714, AND 714X)

Terminal ASCII (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	ASCII Graphic	Control Character††	Octal Code†††	ASCII Graphic	Control Character
345	e		145	e	
346	f		146	f	
351	i		151	i	
352	j		152	j	
354	l		154	l	
357	o		157	o	
361	q		161	q	
362	r		162	r	
364	t		164	t	
367	w		167	w	
370	x		170	x	
373	{		173	{	
374	¦ or ↑ or |		174	¦	
375	}		175	}	
376	~ or ¬		176	~	

†Shown with odd parity, which is the default for these terminal classes (unless PA=N, an application program receives the same code as in normalized mode).

††A circle around a character indicates that the character key is pressed in conjunction with a CTL, CTRL, CNTRL, or CONTROL key to generate the code.

†††Shown with zero parity (eighth or uppermost bit is always zero).

TABLE A-16. ASCII CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal EBCD (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
000	space		040	space	
001	_ or -		137 or 055	_ or -	
002	¢ or @		140 or 100	` or @	
003	+ or &		053 or 046	+ or &	
004	* or 8		052 or 070	* or 8	
005	Q or q		121 or 161	Q or q	
006	Y or y		131 or 171	Y or y	
007	H or h		110 or 150	H or h	
010	: or 4		072 or 064	: or 4	
011	M or m		115 or 155	M or m	
012	U or u		125 or 165	U or u	
013	D or d		104 or 144	D or d	
014		PN or PUNCH ON	021		device control 1 (tape on)
015		RES or RESTORE	000		null
016		BY or BYPASS	000		null
017		PF or PUNCH OFF	023		device control 3 (tape off)
020	< or 2		074 or 062	< or 2	
021	K or k		113 or 153	K or k	
022	S or s		123 or 163	S or s	
023	B or b		102 or 142	B or b	
024	) or 0		051 or 060	) or 0	
025		undefined	000		null
026		undefined	000		null
027		undefined	000		null
030	' or 6		041 or 066	' or 6	
031	O or o		117 or 157	O or o	
032	W or w		127 or 167	W or w	
033	F or f		106 or 146	F or f	
034		UCS or UPPERCASE	017		shift in§
035		BS or BACKSPACE	010		backspace
036		EOB	027		end transmission block§
037		LCS or LOWERCASE	016		shift out§
040	= or 1		075 or 061	= or 1	
041	J or j		112 or 152	J or j	
042	? or /		077 or 057	? or /	
043	A or a		101 or 141	A or a	
044	(or 9		050 or 071	(or 9	
045	R or r		122 or 162	R or r	
046	Z or z		132 or 172	Z or z	
047	I or i		111 or 151	I or i	
050	% or 5		045 or 065	% or 5	
051	N or n		116 or 156	N or n	
052	V or v		126 or 166	V or v	
053	E or e		105 or 145	E or e	
054		RO or READER STOP	000		null
055		NL or CR or RETURN	015		carriage return
056		LF or LINE FEED	012		linefeed
057		HT or TAB	006		horizontal tabulate
060	; or 3		073 or 063	; or 3	
061	L or 1		114 or 154	L or 1	
062	T or t		124 or 164	T or t	
063	C or c		103 or 143	C or c	
064	" or #		042 or 043	" or #	
065	! or $		041 or 044	! or $	
066	| or ,		174 or 054	¦ or ,	
067	¬ or .		136 or 056	^ or .	

TABLE A-16. ASCII CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal EBCD (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
070	> or 7		076 or 067	> or 7	
071	P or p		120 or 160	P or p	
072	X or x		130 or 170	X or x	
073	G or g		107 or 147	G or g	
074		EOT	004		end of transmission§
075		IL or IDLE or NULL	000		null
076		PRE or PREFIX	001		start of header§
077		DEL	177		delete
100	space		040	space	
101	¯ or -		137 or 055	¯ or -	
102	¢ or @		140 or 100	` or @	
103	+ or &		053 or 046	+ or &	
104	* or 8		052 or 070	* or 8	
105	Q or q		121 or 161	Q or q	
106	Y or y		131 or 171	Y or y	
107	H or h		110 or 150	H or h	
110	: or 4		072 or 064	: or 4	
111	M or m		115 or 155	M or m	
112	U or u		125 or 165	U or u	
113	D or d		104 or 144	D or d	
114		PN or PUNCH ON	021		device control 1 (tape on)
115		RES or RESTORE	000		null
116		BY or BYPASS	000		null
117		PF or PUNCH OFF	023		device control 3 (tape off)
120	< or 2		074 or 062	< or 2	
121	K or k		113 or 153	K or k	
122	S or s		123 or 163	S or s	
123	B or b		102 or 142	B or b	
124	) or 0		051 or 060	) or 0	
125		undefined	000		null
126		undefined	000		null
127		undefined	000		null
130	' or 6		041 or 066	' or 6	
131	O or o		117 or 157	O or o	
132	W or w		127 or 167	W or w	
133	F or f		106 or 146	F or f	
134		UCS or UPPERCASE	017		shift in§
135		BS or BACKSPACE	010		backspace
136		EOB	027		end transmission block§
137		LCS or LOWERCASE	016		shift out§
140	= or 1		075 or 061	= or 1	
141	J or j		112 or 152	J or j	
142	? or /		077 or 057	? or /	
143	A or a		101 or 141	A or a	
144	(or 9		050 or 071	(or 9	
145	R or r		122 or 162	R or r	
146	Z or z		132 or 172	Z or z	
147	I or i		111 or 151	I or i	
150	% or 5		045 or 065	% or 5	
151	N or n		116 or 156	N or n	
152	V or v		126 or 166	V or v	
153	E or e		105 or 145	E or e	
154		RO or READER STOP	000		null
155		NL or CR or RETURN	015		carriage return
156		LF or LINE FEED	012		linefeed
157		HT or TAB	006		horizontal tabulate

TABLE A-16. ASCII CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal EBCD (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
160	; or 3		073 or 063	; or 3	
161	L or 1		114 or 154	L or l	
162	T or t		124 or 164	T or t	
163	C or c		103 or 143	C or c	
164	" or #		042 or 043	" or #	
165	! or $		041 or 044	! or $	
166	| or ,		174 or 054	¦ or ,	
167	¬ or .		136 or 056	^ or .	
170	> or 7		076 or 067	> or 7	
171	P or p		120 or 160	P or p	
172	X or x		130 or 170	X or x	
173	G or g		107 or 147	G or g	
174		EOT	004		end of transmission§
175		IL or IDLE or NULL	000		null
176		PRE or PREFIX	001		start of header§
177		DEL	177		delete
000	space§§		133 thru 135	[or \ or]	
000	space§§		140	`	
000	space§§		173	{	
000	space§§		175 or 176	} or ~	
175		IL or IDLE or NULL§§	002		start of text
175		IL or IDLE or NULL§§	003		end of text
175		IL or IDLE or NULL§§	005		enquire
175		IL or IDLE or NULL§§	007		bell
175		IL or IDLE or NULL§§	013 or 014		vertical tabulate or formfeed
175		IL or IDLE or NULL§§	020		data link escape
175		IL or IDLE or NULL§§	022		device control 2
175		IL or IDLE or NULL§§	024 thru 026		device control 4, negative acknowledge, or synchronize
175		IL or IDLE or NULL§§	030 thru 037		cancel, end of media, substitute, escape, file separator, group separator, record separator, or unit separator

†Shown with odd and even parity; odd parity is the default for this terminal class. (Unless PA=N, the application program receives the same code as in normalized mode.)

††Each input line is assumed to begin in lowercase. Input characters are translated to lowercase ASCII characters unless prefixed by the UCS code. Once a case shift occurs, it remains in effect until another case shift code is received, the page width is reached, or the line is transmitted to the host computer. During output, case is preserved by insertion of case shift codes where needed.

†††Shown with zero parity (eighth or uppermost bit is always zero).

§Not transmitted to the host computer after translation during input.

§§Output translation only.

TABLE A-17. APL CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal EBCD-APL (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD-APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
000	space		040	space	
001	_ or +		137 or 053	_ or +	
002	→ or ←		161 or 160	→ or ←	
003	÷ or X		045 or 146	÷ or X	
004	≠ or 8		042 or 070	≠ or 8	
005	? or Q		077 or 121	? or Q	
006	↑ or Y		171 or 131	↑ or Y	
007	Δ or H		150 or 110	Δ or H	
010	≤ or 4		100 or 064	≤ or 4	
011	| or M		174 or 115	| or M	
012	↓ or U		165 or 125	↓ or U	
013	⌊ or D		144 or 104	⌊ or D	
014		undefined	000		null
015		undefined	000		null
016		undefined	000		null
017		undefined	000		null
020	- or 2		055 or 062	- or 2	
021	⊣ or K		153 or 113	⊣ or K	
022	⌐ or S		163 or 123	⌐ or S	
023	⊥ or B		142 or 102	⊥ or B	
024	^ or 0		046 or 060	^ or 0	
025		undefined	000		null
026		undefined	000		null
027		undefined	000		null
030	≥ or 6		174 or 066	≥ or 6	
031	○ or 0		157 or 117	○ or 0	
032	ω or W		167 or 127	ω or W	
033	¯ or F		136 or 106	¯ or F	
034		UCS or UPPERCASE	017		shift in§
035		BS or BACKSPACE	010		backspace
036		EOB	027		end transmission block§
037		LCS or LOWERCASE	016		shift out§
040	" or 1		042 or 061	" or 1	
041	∘ or J		152 or 112	∘ or J	
042	\ or /		134 or 057	\ or /	
043	∝ or A		141 or 101	∝ or A	
044	∨ or 9		041 or 071	∨ or 9	
045	ρ or R		162 or 122	ρ or R	
046	⊂ or Z		172 or 132	⊂ or Z	
047	⍳ or I		151 or 111	⍳ or I	
050	= or 5		075 or 065	= or 5	
051	τ or N		156 or 116	τ or N	
052	U or V		166 or 126	U or V	
053	∈ or E		145 or 105	∈ or E	
054		undefined	000		null
055		NL or CR or RETURN	015		carriage return
056		LF or LINE FEED	012		line feed
057		HT or TAB	006		horizontal tabulate
060	< or 3		074 or 063	< or 3	
061	□ or L		154 or 114	□ or L	
062	~ or T		164 or 124	~ or T	
063	∩ or C		143 or 103	∩ or C	
064	) or]		051 or 135	) or]	
065	(or [		050 or 133	(or [	
066	; or ,		073 or 054	; or ,	
067	: or .		072 or 056	: or .	
070	> or 7		076 or 067	> or 7	
071	* or P		052 or 120	* or P	
072	⊃ or X		170 or 130	⊃ or X	
073	▽ or G		147 or 107	▽ or G	
074		EOT	004		end of transmission§
075		IL or IDLE or NULL	000		null

TABLE A-17. APL CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal EBCD-APL (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD-APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
076		PRE or PREFIX	001		start of header§
077		DEL	177		delete
100	space		040	space	
101	_ or +		137 or 053	_ or +	
102	→ or ←		161 or 160	→ or ←	
103	÷ or X		045 or 146	÷ or X	
104	≠ or 8		042 or 070	≠ or 8	
105	? or Q		077 or 121	? or Q	
106	↑ or Y		171 or 131	↑ or Y	
107	Δ or H		150 or 110	Δ or H	
110	≤ or 4		100 or 064	≤ or 4	
111	⌈ or M		174 or 115	⌈ or M	
112	↓ or U		165 or 125	↓ or U	
113	⌊ or D		144 or 104	⌊ or D	
114		undefined	000		null
115		undefined	000		null
116		undefined	000		null
117		undefined	000		null
120	- or 2		055 or 062	- or 2	
121	⊣ or K		153 or 113	⊣ or K	
122	⌐ or S		163 or 123	⌐ or S	
123	⊥ or B		142 or 102	⊥ or B	
124	^ or 0		046 or 060	^ or 0	
125		undefined	000		null
126		undefined	000		null
127		undefined	000		null
130	≥ or 6		174 or 066	≥ or 6	
131	○ or O		157 or 117	○ or O	
132	⍵ or W		167 or 127	⍵ or W	
133	⍱ or F		136 or 106	⍱ or F	
134		UCS or UPPERCASE	017		shift in§
135		BS or BACKSPACE	010		backspace
136		EOB	027		end transmission block§
137		LCS or LOWERCASE	016		shift out§
140	" or 1		042 or 061	" or 1	
141	° or J		152 or 112	° or J	
142	\ or /		134 or 057	\ or /	
143	⍺ or A		141 or 101	⍺ or A	
144	⍲ or 9		041 or 071	⍲ or 9	
145	⍴ or R		162 or 122	⍴ or R	
146	⊂ or Z		172 or 132	⊂ or Z	
147	⍳ or I		151 or 111	⍳ or I	
150	= or 5		075 or 065	= or 5	
151	⍵ or N		156 or 116	⍵ or N	
152	U or V		166 or 126	U or V	
153	∈ or E		145 or 105	∈ or E	
154		undefined	000		null
155		NL or CR or RETURN	015		carriage return
156		LF or LINE FEED	012		line feed
157		HT or TAB	006		horizontal tabulate
160	< or 3		074 or 063	< or 3	
161	⍞ or L		154 or 114	⍞ or L	
162	~ or T		164 or 124	~ or T	
163	∩ or C		143 or 103	∩ or C	
164	) or]		051 or 135	) or]	
165	(or [		050 or 133	(or [	
166	; or ,		073 or 054	; or ,	
167	: or .		072 or 056	: or .	
170	> or 7		076 or 067	> or 7	
171	* or P		052 or 120	* or P	
172	⊃ or X		170 or 130	⊃ or X	
173	∇ or G		147 or 107	∇ or G	
174		EOT	004		end of transmission§

TABLE A-17. APL CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal EBCD-APL (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	EBCD-APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
175		IL or IDLE or NULL	000		null
176		PRE or PREFIX	001		start of header§
177		DEL	177		delete
000	space§§		047	'	
000	space§§		140	◇	
000	space§§		173	{	
000	space§§		175	}	
175		IL or IDLE or NULL§§	002		start of text
175		IL or IDLE or NULL§§	003		end of text
175		IL or IDLE or NULL§§	005		enquire
175		IL or IDLE or NULL§§	007		bell
175		IL or IDLE or NULL§§	013 or 014		vertical tabulate or form feed
175		IL or IDLE or NULL§§	020 thru 026		data link escape, device control 1 thru device control 4, negative acknowledge, or synchronize
175		IL or IDLE or NULL§§	030 thru 037		cancel, end of media, substitute, escape, file separator, group separator, record separator, or unit separator

†Shown with odd and even parity; odd parity is the default for this terminal class. (Unless PA=N, the application program receives the same code as in normalized mode.)

††Each input line is assumed to begin in lowercase. Input characters are translated to lowercase ASCII characters unless prefixed by the UCS code. Once a case shift occurs, it remains in effect until another case shift code is received, the page width is reached, or the line is transmitted to the host computer. During output, case is preserved by insertion of case shift codes where needed.

†††Shown with zero parity (eighth or uppermost bit is always zero).

§Not transmitted to the host computer after translation during input.

§§Output translation only.

TABLE A-18. ASCII CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
000	space		040	space	
001	1/4 or 1/2		137 or 135	[or]	
002	T or t		124 or 164	T or t	
003	J or j		112 or 152	J or j	
004	$ or 4		044 or 064	$ or 4	
005	O or o		117 or 157	O or o	
006	L or 1		114 or 154	L or 1	
007	? or /		077 or 057	? or /	
010	% or 5		045 or 065	% or 5	
011	" or '		042 or 041	" or '	
012	E or e		105 or 145	E or e	
013	P or p		120 or 160	P or p	
014		PN or PUNCH ON	021		device control 1 (tape on)
015		RES or RESTORE	000		null
016		BY or BYPASS	000		null
017		PF or PUNCH OFF	023		device control 3 (tape off)
020	@ or 2		100 or 062	@ or 2	
021	.		056	.	
022	N or n		116 or 156	N or n	
023	+ or =		053 or 075	+ or =	
024	Z or z		132 or 172	Z or z	
025		undefined	000		null
026		undefined	000		null
027		undefined	000		null
030	¢ or 6		041 or 066	! or 6	
031	I or i		111 or 151	I or i	
032	K or k		113 or 153	K or k	
033	Q or q		121 or 161	Q or q	
034		UCS or UPPERCASE	017		shift in§
035		BS or BACKSPACE	010		backspace
036		EOB	027		end transmission block§
037		LCS or LOWERCASE	016		shift out§
040	± or 1		174 or 061	¦ or 1	
041	M or m		115 or 155	M or m	
042	X or x		130 or 170	X or x	
043	G or g		107 or 147	G or g	
044	) or 0		051 or 060	) or 0	
045	S or s		123 or 163	S or s	
046	H or h		110 or 150	H or h	
047	Y or y		131 or 171	Y or y	
050	& or 7		046 or 067	& or 7	
051	R or r		122 or 162	R or r	
052	D or d		104 or 144	D or d	
053	: or ;		072 or 073	: or ;	
054		RO or READER STOP	000		null
055		NL or CR or RETURN	015		carriage return
056		LF or LINE FEED	012		linefeed
057		HT or TAB	006		horizontal tabulate
060	# or 3		043 or 063	# or 3	
061	V or v		126 or 166	V or v	
062	U or u		125 or 165	U or u	
063	F or f		106 or 146	F or f	
064	(or 9		050 or 071	(or 9	
065	W or w		127 or 167	W or w	
066	B or b		102 or 142	B or b	
067	_ or -		137 or 055	_ or -	

TABLE A-18. ASCII CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
070	* or 8		052 or 070	* or 8	
071	A or a		101 or 141	A or a	
072	C or c		103 or 143	C or c	
073	,		054	,	
074		EOT	004		end of transmission§
075		IL or IDLE or NULL	000		null
076		PRE or PREFIX	033		escape
077		DEL	177		delete
100	space		040	space	
101	1/4 or 1/2		133 or 135	[or]	
102	T or t		124 or 164	T or t	
103	J or j		112 or 152	J or j	
104	$ or 4		044 or 064	$ or 4	
105	O or o		117 or 157	O or o	
106	L or l		114 or 154	L or l	
107	? or /		077 or 057	? or /	
110	% or 5		045 or 065	% or 5	
111	" or '		042 or 041	" or '	
112	E or e		105 or 145	E or e	
113	P or p		120 or 160	P or p	
114		PN or PUNCH ON	021		device control 1 (tape on)
115		RES or RESTORE	000		null
116		BY or BYPASS	000		null
117		PF or PUNCH OFF	023		device control 3 (tape off)
120	@ or 2		100 or 062	@ or 2	
121	.		056	.	
122	N or n		116 or 156	N or n	
123	+ or =		053 or 075	+ or =	
124	Z or z		132 or 172	Z or z	
125		undefined	000		null
126		undefined	000		null
127		undefined	000		null
130	¢ or 6		041 or 066	! or 6	
131	I or i		111 or 151	I or i	
132	K or k		113 or 153	K or k	
133	Q or q		121 or 161	Q or q	
134		UCS or UPPERCASE	017		shift in§
135		BS or BACKSPACE	010		backspace
136		EOB	027		end transmission block§
137		LCS or LOWERCASE	016		shift out§
140	+ or \		174 or 061	+ or \	
141	M or m		115 or 155	M or m	
142	X or x		130 or 170	X or x	
143	G or g		107 or 147	G or g	
144	) or 0		051 or 060	) or 0	
145	S or s		123 or 163	S or s	
146	H or h		110 or 150	H or h	
147	Y or y		131 or 171	Y or y	
150	& or 7		046 or 067	& or 7	
151	R or r		122 or 162	R or r	
152	D or d		104 or 144	D or d	
153	: or ;		072 or 073	: or ;	
154		RO or READER STOP	000		null
155		NL or CR or RETURN	015		carriage return
156		LF or LINE FEED	012		linefeed
157		HT or TAB	006		horizontal tabulate

TABLE A-18. ASCII CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code Graphic††	Control Character	Octal Code†††	ASCII Graphic	Control Character
160	# or 3		043 or 063	# or 3	
161	V or v		126 or 166	V or v	
162	U or u		125 or 165	U or u	
163	F or f		106 or 146	F or f	
164	(or 9		050 or 071	(or 9	
165	W or w		127 or 167	W or w	
166	B or b		102 or 142	B or b	
167	_ or -		137 or 055	_ or -	
170	¯ or 8		052 or 070	* or 8	
171	A or a		101 or 141	A or a	
172	C or c		103 or 143	C or c	
173	,		054	,	
174		EOT	004		end of transmission§
175		IL or IDLE or NULL	000		null
176		PRE or PREFIX	033		escape
177		DEL	177		delete
000	space§§		047	'	
000	space§§		134	\	
000	space§§		136	^	
000	space§§		140	`	
000	space§§		173	{	
000	space§§		175 or 176	} or ~	
175		IL or IDLE or NULL§§	001		start of header
175		IL or IDLE or NULL§§	002		start of text
175		IL or IDLE or NULL§§	003		end of text
175		IL or IDLE or NULL§§	005		enquire
175		IL or IDLE or NULL§§	007		bell
175		IL or IDLE or NULL§§	013 or 014		vertical tabulate or formfeed
175		IL or IDLE or NULL§§	020		data link escape
175		IL or IDLE or NULL§§	022		device control 2
175		IL or IDLE or NULL§§	024 thru 026		device control 4, negative acknowledge, or synchronize
175		IL or IDLE or NULL§§	030 thru 037		cancel, end of media, substitute, file separator, group separator, record separator, or unit separator

†Shown with odd and even parity; odd parity is the default for this terminal class. (Unless PA=N, the application program receives the same code as in normalized mode.)

††Each input line is assumed to begin in lowercase. Input characters are translated to lowercase ASCII characters unless prefixed by the UCS code. Once a case shift occurs, it remains in effect until another case shift code is received, the page width is reached, or the line is transmitted to the host computer. During output, case is preserved by insertion of case shift codes where needed.

†††Shown with zero parity (eighth or uppermost bit is always zero).

§Not transmitted to the host computer after translation during input.

§§Output translation only.

TABLE A-19. APL CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
000	space		040	space	
001	→ or ←		161 or 160	→ or ←	
002	~ or T		164 or 124	~ or T	
003	. or J		056 or 112	. or J	
004	≤ or 4		100 or 064	≤ or 4	
005	○ or 0		157 or 117	○ or 0	
006	□ or L		154 or 114	□ or L	
007	\ or /		134 or 057	\ or /	
010	= or 5		075 or 065	= or 5	
011	) or]		051 or 035	) or]	
012	∈ or E		145 or 105	∈ or E	
013	* or P		052 or 120	* or P	
014		undefined	000		null
015		undefined	000		null
016		undefined	000		null
017		undefined	023		null
020	¯ or 2		136 or 062	¯ or 2	
021	: or .		072 or 056	: or .	
022	⍑ or N		156 or 116	⍑ or N	
023	+ or X		045 or 146	+ or X	
024	⊂ or Z		172 or 132	⊂ or Z	
025		undefined	000		null
026		undefined	000		null
027		undefined	000		null
030	≥ or 6		174 or 066	≥ or 6	
031	⍳ or I		151 or 111	⍳ or I	
032	⊣ or K		153 or 113	⊣ or K	
033	? or Q		077 or 121	? or Q	
034		UCS or UPPERCASE	017		shift in§
035		BS or BACKSPACE	010		backspace
036		EOB	027		end transmission block§
037		LCS or LOWERCASE	016		shift out§
040	" or 1		042 or 061	" or 1	
041	| or M		174 or 115	| or M	
042	⊃ or X		170 or 130	⊃ or X	
043	∇ or G		147 or 107	∇ or G	
044	∧ or O		045 or 060	∧ or O	
045	⌐ or S		163 or 123	⌐ or S	
046	Δ or H		150 or 110	Δ or H	
047	↑ or Y		171 or 131	↑ or Y	
050	> or 7		076 or 067	> or 7	
051	⍴ or R		162 or 122	⍴ or R	
052	⌊ or D		144 or 104	⌊ or D	
053	(or [		050 or 133	(or [	
054		undefined	000		null
055		NL or CR or RETURN	015		carriage return
056		LF or LINE FEED	012		linefeed
057		HT or TAB	006		horizontal tabulate
060	< or 3		074 or 063	< or 3	
061	U or V		166 or 126	U or V	
062	↓ or U		165 or 125	↓ or U	
063	_ or F		137 or 106	_ or F	
064	⍱ or 9		041 or 071	⍱ or 9	
065	⍵ or W		167 or 127	⍵ or W	
066	⊥ or B		142 or 102	⊥ or B	
067	- or +		055 or 053	- or +	

TABLE A-19. APL CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
070	≠ or 8		042 or 070	≠ or 8	
071	∝ or A		141 or 101	∝ or A	
072	∩ or C		143 or 103	∩ or C	
073	; or ,		073 or 054	; or ,	
074		EOT	004		end of transmission§
075		IL or IDLE or NULL	000		null
076		PRE or PREFIX	033		escape
077		DEL	177		delete
100	space		040	space	
101	→ or ←		161 or 160	→ or ←	
102	~ or T		164 or 124	~ or T	
103	. or J		056 or 112	. or J	
104	≤ or 4		100 or 064	≤ or 4	
105	○ or 0		157 or 117	○ or 0	
106	□ or L		154 or 114	□ or L	
107	\ or /		134 or 057	\ or /	
110	= or 5		075 or 065	= or 5	
111	) or]		051 or 035	) or]	
112	∈ or E		145 or 105	∈ or E	
113	* or P		052 or 120	* or P	
114		undefined	000		null
115		undefined	000		null
116		undefined	000		null
117		undefined	023		null
120	¯ or 2		136 or 062	¯ or 2	
121	: or .		072 or 056	: or .	
122	⍴ or N		156 or 116	⍴ or N	
123	+ or X		045 or 146	+ or X	
124	⊂ or Z		172 or 132	⊂ or Z	
125		undefined	000		null
126		undefined	000		null
127		undefined	000		null
130	≥ or 6		174 or 066	≥ or 6	
131	⍳ or I		151 or 111	⍳ or I	
132	⊣ or K		153 or 113	⊣ or K	
133	? or Q		077 or 121	? or Q	
134		UCS or UPPERCASE	017		shift in§
135		BS or BACKSPACE	010		backspace
136		EOB	027		end transmission block§
137		LCS or LOWERCASE	016		shift out§
140	" or 1		042 or 061	" or 1	
141	| or M		174 or 115	| or M	
142	⊃ or X		170 or 130	⊃ or X	
143	∇ or G		147 or 107	∇ or G	
144	^ or 0		045 or 060	^ or 0	
145	⌈ or S		163 or 123	⌈ or S	
146	Δ or H		150 or 110	Δ or H	
147	⍎ or Y		171 or 131	⍎ or Y	
150	> or 7		076 or 067	> or 7	
151	⍴ or R		162 or 122	⍴ or R	
152	⌊ or D		144 or 104	⌊ or D	
153	(or [		050 or 133	(or [	
154		undefined	000		null
155		NL or CR or RETURN	015		carriage return
156		LF or LINE FEED	012		linefeed
157		HT or TAB	006		horizontal tabulate

TABLE A-19. APL CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (Contd)
(2741)

Terminal Correspondence Code (Transparent Mode Use)			Network ASCII (Normalized Mode)		
Octal Code†	Correspondence Code APL Graphic††	Control Character	Octal Code†††	ASCII-APL Graphic	Control Character
160	< or 3		074 or 063	< or 3	
161	U or V		166 or 126	U or V	
162	↓ or U		165 or 125	↓ or U	
163	_ or F		137 or 106	_ or F	
164	∨ or 9		041 or 071	∨ or 9	
165	ω or W		167 or 127	ω or W	
166	⊥ or B		142 or 102	⊥ or B	
167	- or +		055 or 053	- or +	
170	≠ or 8		042 or 070	≠ or 8	
171	∝ or A		141 or 101	∝ or A	
172	∩ or C		143 or 103	∩ or C	
173	; or ,		073 or 054	; or ,	
174		EOT	004		end of transmission§
175		IL or IDLE or NULL	000		null
176		PRE or PREFIX	033		escape
177		DEL	177		delete
000	space§§		047	'	
000	space§§		140	◇	
000	space§§		173	{	
000	space§§		175 or 176	} or ⊢	
175		IL or IDLE or NULL§§	001		start of header
175		IL or IDLE or NULL§§	002		start of text
175		IL or IDLE or NULL§§	003		end of text
175		IL or IDLE or NULL§§	005		enquire
175		IL or IDLE or NULL§§	007		bell
175		IL or IDLE or NULL§§	013 or 014		vertical tabulate or formfeed
175		IL or IDLE or NULL§§	020		data link escape
175		IL or IDLE or NULL§§	022		device control 2
175		IL or IDLE or NULL§§	024 thru 026		device control 4, negative acknowledge, or synchronize
175		IL or IDLE or NULL§§	030 thru 037		cancel, end of media, substitute, file separator, group separator, record separator, or unit separator

†Shown with odd and even parity; odd parity is the default for this terminal class. (Unless PA=N, the application program receives the same code as in normalized mode.)

††Each input line is assumed to begin in lowercase. Input characters are translated to lowercase ASCII characters unless prefixed by the UCS code. Once a case shift occurs, it remains in effect until another case shift code is received, the page width is reached, or the line is transmitted to the host computer. During output, case is preserved by insertion of case shift codes where needed.

†††Shown with zero parity (eighth or uppermost bit is always zero).

§Not transmitted to the host computer after translation during input.

§§Output translation only.

DIAGNOSTIC MESSAGES B

This appendix lists the following categories of messages affecting network programming:

Execution errors

Assembly errors

Postprocessor errors

Communications Control Program errors

EXECUTION ERRORS

When the Network Access Method's execution time code detects a fatal error, a diagnostic message is written in the application program's dayfile. The diagnostic messages issued by NIP are listed alphabetically in table B-1.

All fatal errors detected by NIP cause the application program to abort without the ability to reprieve itself from the abort. All fatal errors detected by AIP cause the application program to abort and permit the application to reprieve itself from the abort, but no further AIP calls are allowed after the abort occurs.

The form of diagnostic message used by AIP and/or QTRM is partially determined by the library used to provide the routines for the execution run. If the routines are loaded from library NETIO, the only fatal diagnostic issued is:

NETWORK APPLICATION ABORTED, RC=rc.

where rc is a reason code from 01 through 99, with the significance indicated in table B-2. If the AIP and QTRM routines are loaded from library NETIOD, the same fatal diagnostic message is issued, but a supplementary message explaining the reason code is issued, as shown in the Message column of table B-2. The supplementary message begins with the name of the routine that detected the error.

The additional informative message:

NAM VER. x.y - level

is always issued at AIP NETON call processing completion. The numbers x,y, and level, respectively, indicate the version number, variant, and PSR level of the AIP code used.

ASSEMBLY ERRORS

When an application program uses the COMPASS macro version of the AIP calls, the assembly listing can contain the fatal error messages listed in table B-3. These messages are described in detail in section 5.

POSTPROCESSOR ERRORS

The debug log file postprocessor (DLFP) is used to process debug log files. During this processing it can issue the messages shown in table B-4.

COMMUNICATIONS CONTROL PROGRAM ERRORS

The diagnostic messages issued by the Communications Control Program to the terminal operator are listed alphabetically in table B-5.

TABLE B-1. APPLICATION PROGRAM DAYFILE NIP DIAGNOSTIC MESSAGES

Message	Significance	Action	Issued By
ADDRESS OUT OF RANGE	The application program specified an address of 0, 1, or a word outside of its field length on a NETPUT or NETGET type AIP call, or an AIP bug exists.	Change the address and rerun the job. If an incorrect address cannot be found, contact a system analyst; a bug exists in AIP.	NIP
APP WORK LIST ADDR=0	AIP has indicated that NIP should write its reply worklist at address 0. NIP cannot use this address. Either an AIP bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP

TABLE B-1. APPLICATION PROGRAM DAYFILE NIP DIAGNOSTIC MESSAGES (Contd)

Message	Significance	Action	Issued By
BAD AIP OPCODE	AIP has passed an invalid operation code in a worklist sent to NIP. Either an AIP bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP
BAD WORD/ ENTRY COUNT	The number of words or entries in a worklist passed from AIP to NIP exceeded the maximum number permitted. Either an AIP bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP
EXTRA WORKLIST	AIP passed a new worklist to NIP while NIP was still processing a previous worklist. Either an AIP bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP
ILLOGICAL WORKLIST	AIP has passed a worklist to NIP that contains more than one NETWAIT or NETGET request. Either an AIP bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP
INVALID APPLICATION NAME ON NETON	The program attempted to access the network with an aname parameter that does not appear in the local configuration file.	Correct the aname parameter and rerun the job. Check that the local configuration file has been updated to include the application's name.	NIP
INVALID MINACN/ MAXACN ON NETON	One or both of the indicated parameters was out of the range permitted for the installation.	Change the parameters and rerun the job.	NIP
NONEXISTENT APPLICATION ID	NIP has no table entry corresponding to the process number AIP has passed to it to identify the application program. Either an AIP or NAM bug exists, or the application program has bypassed or destroyed its copy of AIP.	Follow site-defined procedure to report and correct product or system problems.	NIP
NOT YET NETTED ON	The application program attempted to use the network's resources before issuing a NETON call. If this message does not occur with the corresponding AIP message, either a bug exists in AIP, or the application program has bypassed or destroyed its copy of AIP.	Change the program and rerun the job.	NIP
SECURITY VIOLATION	The application program has attempted to call NETON as a supervisory or validation program.	Change the program and rerun the job.	NIP

TABLE B-2. APPLICATION PROGRAM DAYFILE AIP AND QTRM DIAGNOSTIC MESSAGES

Reason Code	Message	Significance	Action	Issued By
01 thru 29		Reserved by CDC.		
30	NETON: DUPLICATE NETON REQUEST	The application program has called NETON twice.	Change and rerun the job.	AIP
31	NP$GET: REQUEST INVALID BEFORE NETON	The application program issued a GET-type call before it issued a NETON call, or after it issued a NETOFF call.	Change the program and rerun the job.	AIP
32	NP$PUT: REQUEST INVALID BEFORE NETON	The application program issued a PUT-type call before it issued a NETON call, or after it issued a NETOFF call.	Change the program and rerun the job.	AIP
33	NETWAIT: REQUEST INVALID BEFORE NETON	The application program issued the indicated call before it issued a NETON call, or after it issued a NETOFF call.	Change the program and rerun the job.	AIP
34	NETDBG: REQUEST INVALID BEFORE NETON	The application program issued the indicated call before it issued a NETON call, or after it issued a NETOFF call.	Change the program and rerun the job.	AIP
35 thru 39		Reserved by CDC.		
40	NETON: PREVIOUS REQUEST INCOMPLETE	An AIP call other than to NETOFF or NETCHEK cannot be made while the program is in parallel processing mode and a previous AIP call has not been completed.	Relocate the improperly placed NETON call and rerun the job.	AIP
41	NETSETP: PREVIOUS REQUEST INCOMPLETE	An AIP call other than to NETOFF or NETCHEK cannot be made while the program is in parallel processing mode and a previous AIP call has not been completed.	Relocate the improperly placed NETSETP call and rerun the job.	AIP
42	NP$GET: PREVIOUS REQUEST INCOMPLETE	An AIP call other than to NETOFF or NETCHEK cannot be made while the program is in parallel processing mode and a previous AIP call has not been completed.	Relocate the improperly placed GET-type call and rerun the job.	AIP
43	NP$PUT: PREVIOUS REQUEST INCOMPLETE	An AIP call other than to NETOFF or NETCHEK cannot be made while the program is in parallel processing mode and a previous AIP call has not been completed.	Relocate the improperly placed PUT-type call and rerun the job.	AIP

TABLE B-2. APPLICATION PROGRAM DAYFILE AIP AND QTRM DIAGNOSTIC MESSAGES (Contd)

Reason Code	Message	Significance	Action	Issued By
44	NETWAIT: PREVIOUS REQUEST INCOMPLETE	An AIP call other than to NETOFF or NETCHEK cannot be made while the program is in parallel processing mode and a previous AIP call has not been completed.	Relocate the improperly placed NETWAIT call and rerun the job.	AIP
45 thru 49		Reserved by CDC.		
50	NP$ON: INVALID PROCESS NUMBER	A bug exists in the operating system or NAM. The process number assigned to the application program during processing of its NETON call was out of range.	Follow site-defined procedure to report and correct product or system problems.	AIP
51	NP$XFER: NWL HAS OVERFLOWED	The debug option code in AIP detected an error condition not caused by an application program AIP call.	Follow site-defined procedure to report and correct product or system problems.	AIP
52 thru 66		Reserved by CDC.		
67	NP$XFER: NIP NOT AVAILABLE AT A SCP	The application program reprieved itself after being aborted, but NIP has also aborted. The only AIP call that can be issued after NIP aborts is a NETOFF.	Change the application program reprieve procedure and rerun the job.	AIP
68	FETCH ILLEGAL FIELD MNEMONIC	Either the field or value parameter in the indicated call was not found.	Correct the call and rerun the job.	AIP
69	STORE ILLEGAL FIELD MNEMONIC	Either the field or value parameter in the indicated call was not found.	Correct the call and rerun the job.	AIP
70	QTENDT: REQUEST INVALID BEFORE QTOPEN	A QTENDT call is illegal before a QTOPEN call or after a QTCLOSE call.	Correct the statement sequence and rerun the job.	QTRM
71	QTGET: REQUEST INVALID BEFORE QTOPEN	A QTGET call is illegal before a QTOPEN call or after a QTCLOSE call.	Correct the statement sequence and rerun the job.	QTRM
72	QTPUT: REQUEST INVALID BEFORE QTOPEN	A QTPUT call is illegal before a QTOPEN call or after a QTCLOSE call.	Correct the statement sequence and rerun the job.	QTRM
73	QTLINK: REQUEST INVALID BEFORE QTOPEN	A QTLINK call is illegal before a QTOPEN call or after a QTCLOSE call.	Correct the statement sequence and rerun the job.	QTRM

TABLE B-2. APPLICATION PROGRAM DAYFILE AIP AND QTRM DIAGNOSTIC MESSAGES (Contd)

Reason Code	Message	Significance	Action	Issued By
74	QTTIP: REQUEST INVALID BEFORE QTOPEN	A QTTIP call is illegal before a QTOPEN call or after a QTCLOSE call.	Correct the statement sequence and rerun the job.	QTRM
75 thru 79		Reserved by CDC.		
80	QTOPEN: DUPLICATE QTOPEN	The application program attempted to perform QTOPEN a second time.	Remove the extra QTOPEN statement and rerun the job.	QTRM
81	QTOPEN: NIT NUM-TERMS FIELD IS ZERO	The num-terms field in the network information table was zero when QTOPEN was called.	Correct the table and rerun the job.	QTRM
82	QTOPEN: NETON REJECTED	The application program was not allowed to access the network. Either another application with the same name has accessed the network or the network operator has disabled the application from accessing the network.	Rerun the job after contacting the local operator.	QTRM
83	QTOPEN: NETWORK NOT AVAILABLE	The network is not running or it temporarily does not have enough resources to allow this application to access the network.	Rerun the job later.	QTRM
84 thru 94		Reserved by CDC.		
95	QTLINK: NO SUPPORT-A-TO-A	The application program requested connection to another application program when the support-A-to-A field is not set.	Change the program to set the Support-A-to-A field before the call to QTOPEN and rerun the job.	
96 thru 98		Reserved by CDC.		
99	QTGET: NETWORK LOGICAL ERROR, TYPE n	NAM has sent a logical error supervisory message to the application program; n is the reason code from the logical error supervisory message. The logical error is due to a QTPUT call with bad parameters stored in the network information table.	Correct the parameter fields before issuing the QUPUT call.	QTRM

TABLE B-3. AIP MACRO ASSEMBLY LISTING DIAGNOSTIC MESSAGES

Message	Significance	Action	Issued By
ERR FIRST PARAMETER MISSING	At least one parameter is required in the AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR MUST BE LIST=	A parameter is required after LIST= in the second calling format by the AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR NSUP ADDRESS MISSING	Address of nsup word is not provided in the first or third calling format by the NETON AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR STATUS ADDRESS MISSING	Address of status word is not provided in the first or third calling format by the NETON AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR MINACN ADDRESS MISSING	Address of MINACN word is not provided in the first or third calling format by the NETON AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR MAXACN ADDRESS MISSING	Address of MAXACN word is not provided in the first or third calling format by the NETON AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR HEADER AREA ADDRESS MISSING	Address of application block header is not provided in first or third calling format by the NETGET, NETGETF, NETGETL, or NETGTFL AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR TEXT AREA ADDRESS MISSING	Address of text area is not provided in the first or third calling format by the NETGET, NETGETF, NETGETL or NETGTFL AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR TEXT LIMIT IS MISSING	Address of text limit of block acceptable is not provided in the first or third calling format by the NETGET, NETGETF, NETGETL or NETFTFL AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR SECOND PARAMETER MISSING	Second parameter is not provided in the first or third calling format by the NETPUT, NETREL, NETSETF, NETSTC, NETWAIT, NETPUTF or NETDBG AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR THIRD PARAMETER MISSING	Third parameter is not provided in the first or third calling format by the NETPUTF or NETDBG AIP call that caused the error.	Correct the call and reassemble the job.	AIP
ERR PARAMETER MISSING	The parameter is not provided in the NETSETP AIP call that caused the error.	Correct the call and reassemble the job.	AIP

TABLE B-3. AIP MACRO ASSEMBLY LISTING DIAGNOSTIC MESSAGES (Contd)

Message	Significance	Action	Issued By
ERR field ERROR IN 1ST PARAMETER	The first parameter provided in the NFETCH or NSTORE call that caused the error is not valid. The field parameter indicates the field in which the error occurs.	Correct the call and reassemble the job.	AIP
ERR field ERROR IN FIELD MNEMONICS	The second parameter provided in the NFETCH or NSTORE call that caused the error is not a valid symbolic field name. The field parameter indicates the field in which the error occurs.	Correct the call and reassemble the job.	AIP
ERR field ILLEGAL REGISTER NAME	The third parameter provided in the NFETCH call that caused the error is not a valid register. The field parameter indicates the field in which the error occurs.	Correct the call and reassemble the job.	AIP
ERR field ERROR IN BRD PARAMETER	The third parameter provided in the NSTORE call that caused the error is not a valid register. The field parameter indicates the field in which the error occurs.	Correct the call and reassemble the job.	AIP

TABLE B-4. DLFP DAYFILE, ERROR, AND INFORMATIVE MESSAGES

Message	Significance	Action	Issued By
BAD DEBUG LOG FILE	DLFP did not process the debug log file because the content of the file was bad.	Correct and rerun.	DLFP
BAD DIRECTIVE TABLE ENTRY	DLFP detected an error in its internal tables.	Correct and rerun.	DLFP
DLFP COMPLETE	DLFP completed processing the debug log file, if any.	None.	DLFP
DUPLICATE FILE NAME	The same file name was used on more than one parameter on the control card.	Correct and rerun.	DLFP
EMPTY DEBUG LOG FILE	The debug log file was empty.	None.	DLFP
ERROR IN B DIRECTIVE	B directive is not followed by keyword operator.	Correct and rerun.	DLFP
ERROR IN BD= DIRECTIVE	Date is invalid or missing.	Correct and rerun.	DLFP
ERROR IN BT= DIRECTIVE	Time is invalid or missing.	Correct and rerun.	DLFP
ERROR IN C DIRECTIVE	C directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN CN= DIRECTIVE	Connection number is invalid or missing.	Correct and rerun.	DLFP

TABLE B-4. DLFP DAYFILE, ERROR, AND INFORMATIVE MESSAGES (Contd)

Message	Significance	Action	Issued By
ERROR IN DN= DIRECTIVE	DN directive used incorrectly.	Correct and rerun.	DLFP
ERROR IN E DIRECTIVE	E directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN ED= DIRECTIVE	Date is invalid or missing.	Correct and rerun.	DLFP
ERROR IN ET= DIRECTIVE	Time is invalid or missing.	Correct and rerun.	DLFP
ERROR IN F DIRECTIVE	F directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN LE= DIRECTIVE	Length is an invalid value or missing.	Correct and rerun.	DLFP
ERROR IN N DIRECTIVE	N directive is not followed by a keyword separator.	Correct and rerun.	DLFP
ERROR IN NM= DIRECTIVE	Number is invalid or missing.	Correct and rerun.	DLFP
ERROR IN P DIRECTIVE	P directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN PF= DIRECTIVE	Hexadecimal number is invalid, not two digits, or missing.	Correct and rerun.	DLFP
ERROR IN PS= DIRECTIVE	Hexadecimal number is invalid, not four digits, or missing.	Correct and rerun.	DLFP
ERROR IN R DIRECTIVE	R directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN SM= DIRECTIVE	Number is invalid, or missing.	Correct and rerun.	DLFP
ERROR IN SN= DIRECTIVE	SN directive used incorrectly.	Correct and rerun.	DLFP
ERROR IN T DIRECTIVE	T directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN U DIRECTIVE	U directive is not followed by keyword separator.	Correct and rerun.	DLFP
ERROR IN X DIRECTIVE	X directive is not followed by keyword separator.	Correct and rerun.	DLFP
ILLEGAL CHARACTER	The directive record contains a character that is not a letter, a digit, an equal sign, a comma, or a blank.	Correct and rerun.	DLFP
ILLEGAL FILE NAME	The file name contains characters other than letters and digits or it begins with a number.	Correct and rerun.	DLFP
ILLEGAL PARAMETER	DLFP does not recognize a parameter on the control card.	Correct and rerun.	DLFP
LOG FILE NOT CLOSED	Debug log file was not closed correctly. Either NETOFF or NETREL was not called before the application terminated.	Correct the application program for future executions, if possible.	DLFP
MULTIPLE COMMAS BETWEEN DIRECTIVES	Two or more commas were used with no directive in between them.	Correct and rerun.	DLFP

TABLE B-4. DLFP DAYFILE, ERROR, AND INFORMATIVE MESSAGES (Contd)

Message	Significance	Action	Issued By
NO MESSAGES FOUND	No messages were found with the specified keywords.	None.	DLFP
OVER 10 VALID CHARS BETWEEN KEYWD SEP	The string of valid characters between the keyword separator was greater than 10 characters. A valid character is a letter, a digit, or an equal sign.	Correct and rerun.	DLFP
PARAMETER FORMAT ERROR	A parameter on the control card is not formatted correctly.	Correct and rerun.	DLFP
PARAMETER SPECIFIED TWICE	A parameter on the control card appears more than once.	Correct and rerun.	DLFP
UNRECOGNIZABLE KEYWORD	A nonexistent keyword was used, or the first keyword did not begin in column one.	Correct and rerun.	DLFP

TABLE B-5. CCP USER DIAGNOSTIC MESSAGES

Message	Signifiance	Action	Issued By
FROM NOP... message text	A message from the NPU operator is displayed.	None.	CCP
HOST AVAILABLE	You have selected an available host.	As indicated by the user status message: ENTER INPUT TO CONNECT TO HOST.	CCP
HOST BUSY	The requested connection was rejected by the host.	As indicated by the user status message: ENTER HD TO SEE HOST STATUS.	CCP
HOST CONNECTED	Your terminal has been connected to the host.	As indicated by the user status message: READY FOR INPUT or TERMINAL DISABLED BY NOP.	CCP
HOST DISCONNECTED	The NPU has terminated communication with the host software; connection was not completed.	As indicated by the user status message: ENTER INPUT TO CONNECT TO HOST or TERMINAL DISABLED BY NOP.	CCP
HOST UNAVAILABLE	You have selected and attempted to establish a connection with a host that is currently unavailable.	As indicated by the user status message: ENTER HD TO SEE HOST STATUS.	CCP
INPUT DISCARDED	CCP has discarded user input.	Reenter the input.	CCP
NO HOST AVAILABLE	No hosts are available.	As indicated by the user status message: ENTER HD TO SEE HOST STATUS.	CCP

TABLE B-5. CCP USER DIAGNOSTIC MESSAGES (Contd)

Message	Significance	Action	Issued By
NO HOST CONNECTED	You are not connected to a host.	As indicated by the user status message: ENTER INPUT TO CONNECT TO HOST.	CCP
NO HOST SELECTED	A host is available but you did not select one.	As indicated by the user status message: ENTER HN=nn TO SELECT HOST.	CCP
OVER..	Page wait has occurred. More output is available.	Enter an empty line.	CCP
REPEAT..	System ready to accept input.	Reenter last input.	CCP
WAIT..	Because of a temporary overload condition, a portion of the current input line has been discarded.	Cancel current input line and repeat input after REPEAT.. output.	CCP
xx ACCEPTED..	Informative message that indicates a valid terminal definition command was entered and accepted. xx is the terminal definition command that was accepted.	None.	CCP
xx DUPLICATE CHARACTER	The value you specified in the terminal definition command xx conflicts with another terminal definition.	Try another value.	CCP
xx ERR	The terminal definition command has been rejected because an invalid command syntax was entered. xx is the terminal parameter mnemonic of the terminal definition command.	Reenter the command correctly.	CCP
xx ILLEGAL COMMAND	The characters xx must be a terminal definition mnemonic or interactive status command.	Ensure the accuracy of your entry and retry.	CCP
xx ILLEGAL TERMINAL CLASS	The value you specified in the terminal definition command xx is not valid for your terminal.	Choose an appropriate value (refer to appendix F).	CCP
xx ILLEGAL VALUE	You specified an invalid value in the terminal definition commands xx.	Choose a valid value (refer to appendix F.	CCP
xx VALUE INAPPROPRIATE	The value you specified in the terminal definition command xx is not appropriate for your terminal.	Choose an appropriate value (refer to appendix F).	CCP
DEL	The system has deleted the last line of input. This occurred when you entered the cancel character (CN) followed by the end-of-line/ end-of-block terminator.	None.	CCP

GLOSSARY C

TERMS

This appendix contains terms and mnemonics unique to the description of the software presented in this manual. It also contains terms whose interpretation within this manual is intended to be more constrained or different from that commonly made. Some terms used in other manuals for the network software are included for the reader's convenience when reconciling terminology.

Acknowledgment, Block -
A message returned to the sender confirming the delivery of one block; referred to as BACK in CCP documentation.

Address -
A location of data (as in the main or micro NPU memory) or of a device (as a peripheral device or terminal).

APL -
A scientific programming language characterized by powerful operators defined as single symbols.

Application Block Header (ABH) -
A single 60-bit word description accompanying every block passing between an application program and NAM.

Application Block Limit (ABL) -
The number of unacknowledged blocks a logical connection is allowed to have outstanding (queued by the network) at any one time.

Application Block Number (ABN) -
A field in the application block header. An application-assigned number used to identify a particular data message block.

Application Block Type (ABT) -
A field in the application block header defining the accompanying block as either data or supervisory, null or not null, and indicating which block is the last block of a message.

Application Character Type (ACT) -
A field in the application block header defining the byte size and packing of text characters.

Application Connection Number (ACN) -
A number assigned by NAM to identify a particular logical connection within an application program.

Application Interface Program (AIP) -
A group of routines that reside in the application program's field length. These routines buffer communication between the application program and the network, using the system control point feature of NOS.

Application List Number (ALN) -
An application-program-assigned number used to identify a particular group of logical connections belonging to the application program.

Application Name (ANAME) -
Up to seven 6-bit letters or digits (the first must be a letter) used to identify an application program. It is used by another application program, by a terminal operator when connection to the application is requested, and by the host operator to give commands.

Application Program -
A program resident in a host computer that provides an information storage, retrieval, and/or processing service via the data communication network and the Network Access Method. Application programs always use the system control point feature of NOS to communicate with the Network Access Method. In the context of network software, an application program is not an interactive job, but rather a terminal servicing facility. A terminal servicing facility provides terminal users with a specific processing capability such as remote job entry from batch terminals, transaction processing, entry and execution of interactive jobs, and so forth. For example, the standard CDC interactive facility IAF makes terminal input and output appear the same to an executing program as file input and output; IAF is a network application program, but the executing program using IAF is an interactive job.

Archetype Terminal -
The specific terminal equipment possessing all of the attributes used as defaults for the definition of one terminal class. Each terminal class has a corresponding archetype terminal.

Asynchronous -
A transmission in which each information character is individually synchronized by the use of start and stop bits. The gap between each character is not necessarily of fixed length.

Asynchronous Protocol -
The protocol used by asynchronous, teletype-like devices. For CCP, the protocol is actually the set of protocols for eight types of real terminals. The NPU/terminal interface is handled by the ASYNC TIP.

Automatic Input -
An output mode that prefixes up to 20 characters of the output message to the input reply.

Automatic Login -
The process whereby one or more of the Network Validation Facility login dialog parameters is supplied to NVF from the local configuration file. Parameters supplied through automatic login configuration of a terminal suppress prompting for the corresponding dialog entries and override any entries made from the terminal.

Automatic Recognition -
The process whereby the Terminal Interface Program identifies characteristics of a terminal when the terminal's communication line becomes

active. The Terminal Interface Program determines sub-TIP type and terminal class (and, for mode 4 terminals, the cluster and terminal addresses) by various methods for lines configured for automatic recognition. The Communications Supervisor then matches these parameters against the descriptions of specific terminals in the network configuration file; the terminal with the closest match to the empirically determined parameters is automatically recognized as the terminal on the communication line.

Bandwidth -
For CCP, bandwidth indicates the transfer rate (in characters per second) between the NPU and the terminals.

Base System Software -
The relatively invariant set of programs in CCP that supplies the monitor, timing, interrupt handling, and multiplexing functions for the NPU. Base software also includes common areas, diagnostics, and debugging utilities.

Batch Device -
A device that is capable of conducting input only or output only operations. A batch device is incapable of performing interactive communications. Card readers, line printers, and plotters are examples of batch devices. Batch devices are sometimes referred to as passive devices.

Batch File Command -
A command from RBF in the host which alters the file characteristics of subsequent data transfers for a batch device. The characteristics which can be changed include: code type, suppressing carriage control on output, changing file limits (maximum size of a file in characters), and whether or not lace cards should be generated for a card punch. For HASP and BSC terminals, 026/029 card type can be changed from a terminal through a request to RBF.

Binary Synchronous Communications (BSC) -
A communications protocol supported by the BSC TIP. This protocol connects IBM 2780 or 3780 terminals to the NPU using half-duplex synchronous transmissions in a point-to-point mode. The terminals have batch devices which use EBCDIC code. Transparent data exchanges are permitted. The terminals are structured to have a virtual console (interactive device). This is composed of a card reader for input and a printer for output.

Block -
In the context of network communications, a portion or all of a message. A message is divided into blocks of one or more words (2 bytes/word in the NPU) to facilitate buffering, transmission, error detection and correction of variable length data streams. Differing block protocols apply to the host/NPU and the NPU/terminal interfaces.

Block Acknowledgment -
See Acknowledgment, Block.

Block Header -
See Application Block Header.

Block Interface Package (BIP) -
A group of modules that provide routing, service message handling, and some common TIP subroutines including assistance in IVT block and PRUB formation for upline messages. By making hold/queue decisions for downline batch messages, the BIP also has some batch data stream flow control capability.

Block Limit -
The number of message blocks that can be awaiting delivery at any one time in either the host-to-NPU direction or the NPU-to-host direction for a single device.

Block Protocol -
The protocol governing block transfers of information between the host and the local NPU. Data is transferred in IVT blocks or PRU blocks (PRUBs).

Block Type -
See Application Block Type.

Break -
A method employed by a terminal operator to interrupt output or input in progress. Also, an element of the block protocol that indicates an interruption of the data stream.

Broadcast Message -
A message generated by the system or by an operator using the system. The message is sent to one (broadcast one) or all (broadcast all) of the consoles in the system.

Buffer Threshold -
The minimum number of buffers available for assignment to new tasks. As the buffer level falls toward the threshold, new tasks are rejected (regulation) within CCP.

Buffering -
The process of collecting data together in buffers. Ordinarily, no action on the data is taken until the buffer is filled. Filled buffers include the case where data is terminated before the end of the buffer and the remaining space is filled with extraneous matter.

Byte -
A group of contiguous bits. Unless prefixed (for example, a 6-bit byte), the term implies 8-bit groups. When used for encoding character data, a byte represents a single character. IVT uses 7-bit ASCII characters with the eight bit reserved for parity.

Cassette -
The magnetic tape device in an NPU used for bootstrap loading of off-line diagnostics and (in remote NPUs) the bootstrap load/dump operation.

CE Error Message -
A message containing information concerning hardware and/or software malfunctions.

Character -
A coded byte of data, such as a 6-bit display code or 7-bit ASCII code. Terminals expect a wide range of codes. Network products are responsible for translating between terminal codes and host codes. Unless otherwise specified, references to characters in this manual are to ASCII 7-bit byte characters.

Character Type -
See Application Character Type.

Circular Input Buffer (CIB) -
This fixed buffer is used by the mux subsystem to collect all data passing upline from the multiplexer. The buffer is controlled by a put pointer for the multiplexer and a pick pointer used to demultiplex data to individual line-oriented data buffers.

Cluster -
Mode 4 devices grouped by a common cluster address.

Cluster Address -
The hardware address of a cluster. This term is used in several ways within mode 4 communications documentation, as shown in table C-1.

TABLE C-1. MODE 4 NOMENCLATURE EQUIVALENCE

Networks Nomenclature	Mode 4A Nomenclature	Mode 4C Nomenclature
Network processing unit	Data source	Control station
Cluster address	Site address	Station address
Cluster controller	Equipment controller	Station
Terminal address	Station address	Device address

Command Driver -
The hardware driver that controls the mux subsystem.

Common Area -
Within CCP, areas of main memory dedicated to system and global data. These are usually below address 1000_{16}.

Communication Element -
Any entity that constitutes a point of input to, or output from, the data communication network. This includes terminal devices, terminals, communication lines, and application programs.

Communication Line -
A complete communication circuit between a terminal and its network processing unit.

Communication Network -
The portion of the total network comprising the linked network processing units. The communication network excludes the host computer and terminals and is approximately equivalent to the set of all network elements configured as part of the total network.

Communications Control Program (CCP) -
A portion of the network software that resides in a 255x Series network processing unit. This set of modules performs the tasks delegated to the NPU in the network message processing system. This software can include such routines as the Terminal Interface Program.

Communications Supervisor (CS) -
A portion of the network software, written as an application program; the Communications Supervisor configures and controls the status of NPUs and all their communication lines and terminals.

Configuration -
See System Configuration.

Connection -
See Logical Connection.

Connection Number (CN) -
A unique number assigned to each active device on a logical link.

Console -
A device devoted to network control processing. An example of a console is the NPU Operator's (NOP) terminal. A console attached to the NPU can be used for offline processing.

Constant Carrier -
A communication line with a transmission carrier signal that remains on continuously; failure is reported if the carrier signal received remains off for a period of time that equals or exceeds a failure verification period.

Contention -
The state that exists in a bidirectional transmission line when both ends of the line try to use the line for transmission at the same time. All protocols contain logic to resolve the contention situation.

Control Blocks -
(1) The types of blocks used to transmit control (as opposed to data) information; (2) Blocks assigned for special configuration/status purposes in the NPU. The major blocks are line control blocks (LCB), logical link control blocks (LLCB), logical channel control blocks (LCCB), terminal control blocks (TCB), queue control blocks (QCB), buffer maintenance control blocks (BCB), mux line control blocks (MLCB), text processing control blocks (TPCB), and diagnostics control blocks (DCB).

Controlled Carrier -
A communication line with a transmission carrier signal that is raised and lowered with each block transmitted; failure is reported if the carrier signal received does not fluctuate in a similar fashion.

Controlled Terminal -
A terminal whose input can be started and stopped by the network software. When a terminal places data on a communication line only in response to a poll, the maximum input rate can be controlled by controlling the polling rate. Mode 4 terminals are controlled.

Coupler -
A hardware module resident in a front-end network processing unit. That coupler links the network processing unit to a host computer. Transmissions across the coupler use block protocol.

Cross -
The software support system for CCP. These programs, which are run on the host, support source code programming in PASCAL, macro-assembler, and microassembler languages. The compiled or assembled output of the Cross programs are in object code format on host computer files (source code is also kept in host files). The object code files are processed by other Cross programs and host installation programs into a downline load file for an NPU.

Cyclic Redundancy Check (CRC) -
A check code transmitted with blocks/frames of data. It is used by several protocols including the HASP and CDCCP protocols.

Data -
Any portion of a message created by the source, exclusive of any information used to accomplish transmission of such a message.

Data Buffer -
A block of 64 contiguous words in CCP used for storing data (2 characters per word). A buffer usually has a header of one or more words. Data within a data buffer is delimited by pointers to the first and last characters (data buffers are character oriented). If the data cannot all fit into one buffer, an additional buffer is assigned and is chained to the current buffer. Buffer assignment continues until the entire message is contained in the chain of buffers. Buffers are chained together only to the forward direction.

Data Compression -
The technique of transmitting a sequence of identical characters as a control character and a number representing the length of the sequence. HASP and BSC protocols support data compression.

Data Set -
A hardware interface which transforms analog data to digital data and the converse.

DDLTs -
Special diagnostic programs which use a highly structured table technique to aid the troubleshooter in isolating a problem.

Debugging -
The process of running a program to rid it of anomalies. CCP supplies debugging aids for programs (TUP, PBTIPDG, and PBDEBUG) and for run-time PASCAL programs (QDEBUG and its associated programs) AIP supplies the application programs with debug log files.

Dedicated Line -
A communication line that is permanently connected between a terminal and a network processing unit. Contrast with Switched Line.

DEFINE -
An NDL statement that provides the macro-like capability of substituting an identifier in coding for a more complex entity. When the coding is processed, the identifier is interpreted as if it had been replaced by the complex entity. Also, a NOS control statement that creates permanent files.

Destination -
The device or application program designated to receive the message.

Destination Node (DN) -
The NPU node that directly interfaces to the destination of a data message block. For instance, the DN of an upline message may be the host process which passes the message to the application program responsible for processing the message.

Device -
A separately addressable portion or all of a terminal. This term is used in various ways within mode 4 communications documentation, as shown in table C-1.

Diagnostics -
Software programs or combinations of programs or tables which aid the troubleshooter in isolating problems.

Direct Access File -
In the context of NOS permanent files, a direct access file is a file that is accessed and modified directly.

Direct Calls -
The method of passing control directly from one program to another. This is the usual transfer mode for CCP. Some CCP calls are indirect, through the monitor. Such OPS level indirect calls pass information to the called program through parameter areas called worklists. See Worklist.

Direct Memory Access (DMA) -
The high-speed input/output channel to the NPU main memory. This channel is used for host/NPU buffered transfers.

Directories -
Tables in CCP which contain information used to route blocks to the proper interface and line. There are directories for source and destination node and for connection number. A routed message is attached to the TCB for the line over which the message will pass.

Downline -
The direction of output information flow, from host to NPU to terminal.

Dump -
In the context of CCP, the process of transferring the contents of the NPU main memory, registers, and file 1 registers to the host. The dump can be processed by the Network Dump Analyzer in the host to produce a listing of the dumped information in hexadecimal format.

Echo -
The process of displaying a keystroke on a terminal's console. Echoing can be done from the TIP, from a modem, or from the terminal itself.

Echoplex -
The process of returning received characters on a full-duplex line. Not all terminals on full-duplex communication lines are capable of echoplex operation.

File -
A unit of batch data. Files are transferred between application programs and terminals by using PRUBS on the NPUs host side and transmission blocks on the NPUs terminal side. A file contains one or more records. Example: a card reader job can consist of a file containing the card image records of all the cards in the job deck.

File Registers -
The two sets of microregisters (file 1 and file 2) in the NPU. File 1 registers contain parameter information that is reloaded whenever the NPU is initialized. Microprograms using file 1 registers may also change values in them. File 2 registers are invariant firmware registers that come preprogrammed with the NPU.

Format Effectors (FE) -
Characters in an output data stream that determine the appearance of data at the console. A format effector usually takes the form of a single character in the output message. For printing devices, the character is translated by the output side of the TIP into a combination of carriage returns, line feeds, or spaces. Similarly, FEs for displays can command new lines, screen clearing, or cursor positioning.

Frame -
A frame is a block of data sent across a high-speed link. It is composed of control bytes, a CRC sum, and (in some cases) data bytes in sub-block sequence. A sub-block can be a block protocol block or a part of a block. The frame is the basic communications unit used in trunk (NPU to NPU) communications and provides high-data density in bit-serial format over data-grade lines, as well as data assurance.

Frames are transmitted as a sequence of bytes through the mux subsystem which uses a hardware-controlled frame on the input and output mux loops.

Free-Wheeling Terminal -
When a terminal can input at the discretion of the terminal user and has an input rate that cannot be controlled directly. Asynchronous terminals are free-wheeling. Contrast with Controlled Terminal.

Front-End NPU -
A network processing unit which directly interfaces to one or more hosts. Synonymous with local NPU.

Full Duplex (FDX) -
Two-way simultaneous transmission on a communication line.

Function Codes -
Codes used by the service module to designate the type of function (command or status) being transmitted. Two codes are defined: Primary Function Code (PFC) and Secondary Function Code (SFC). Function codes are also used between NAM and the application programs in all supervisory messages.

Global Variables -
PASCAL variables which are defined for use by any CCP program. Contrast global variables with local variables, which are identified only within a program.

Half Duplex (HDX) -
Two-way alternating transmission on a communication line. Normally a single set of data lines carry input, output, and part of the control information. Contention for use is possible in HDX mode and must be resolved by the protocol governing line transfers.

Halt Codes -
Codes generated by the NPU when it executes a soft-stop. These codes, which indicate the cause of the stoppage, are contained in a CCP dump.

HASP -
A protocol based on the BSC protocol; it is used by HASP workstations. A workstation has both interactive and batch devices. The standard code of all HASP devices is EBCDIC; however, transparent batch data exchanges with the host are also permitted. The HASP TIP converts interactive HASP data from EBCDIC transmission blocks to ASCII IVT blocks; it converts batch HASP data from EBCDIC transmission blocks to display code PRUBs.

Header -
The portion or portions of a message holding information about the message source, destination, and type. During network movement, a message can acquire several headers. For example, during movement of a message from a terminal to the host over an X.25 network, the message acquires the following headers: one at the terminal (also a trailer), one for the frame, one for the packet, and another for the host block. Headers are discarded by the appropriate stage of processing, so that in this example, the host sees only the host block header. Conversely, headers are generated and discarded as needed downline, so that the terminal sees only the terminal header (and trailer).

Header Area (HA) -
An area, usually one 60-bit word, within the application program containing the application block header for a NETPUT or NETPUTF call, or the area to receive the header for a NETGET, NETGETL, NETGETF, or NETGTFL call.

High-Speed Synchronous Line -
A data transmission line operating at or above 19200 b/s. These lines are normally used for local LIP/remote LIP transfers and for X.25 and HASP network transfers.

Host -
The computer that controls the network and contains the applications programs that process network messages.

Host Interface Package (HIP) -
The CCP program which handles block transfers across the host/local NPU interface. The HIP transfers control blocks and data blocks (IVT blocks or PRUBs).

Host Node -
The node ID number of the NPU coupler that directly interfaces with a host computer.

Host Operator (HOP) -
The operator who resides at the system console, initiates NAM, controls NPUs and network-related host elements. The HOP may do all NPU operator (NOP) functions as well as those functions unique to the HOP despite the existance of NOPs. There can be only one HOP. Contrast with NPU operator.

Identifier (ID) -
Identifiers can refer to ports, nodes, lines, links, or terminals. Any hardware element or connection can have an ID, normally a sequentially assigned number.

Initialization -
The process of loading an NPU and optionally dumping the NPU contents. After downline loading from the host, the NPU network-oriented tables are configured by the host so that all network processors have the same IDs for all network terminals, lines, trunks, etc.

Input -
Information flowing upline from terminal to host computer.

Input Buffer -
In the context of CCP, a data buffer containing a message destined to the host. These buffers are dynamically allocated and released.

Input Parameter -
A parameter in an AIP call that provides input to the AIP routine. An input parameter can be a constant, an expression, or a symbolic address for such values. Input parameters are not altered by the completion of AIP processing.

Interactive Device -
Any device capable of conducting both input and output, making it capable of dialog with the Network Validation Facility. Also known as a console-type device. An interactive terminal device is serviced by an application program using the interactive virtual terminal interface. Contrast with Passive Device.

Interactive Virtual Terminal (IVT) -
A block protocol format for interactive terminal consoles. CCP TIPs convert all upline interactive messages to this format (exception: no transformations are made to transparent data except to put the messages into block format). By this method, application programs in the host need only to be able to process interactive data in IVT format rather than in the multiplicity of formats that real terminals use. Downline messages from the host to interactive devices are converted from IVT to real terminal format. IVT processing is controlled by the TIPs; the TIPs use some common IVT modules.

Interface (NPU) -
The set of hardware and software that permits transfers between the NPU and an external device. There are three principal interfaces: to the host through a coupler (block protocol in IVT or PRU format handled by a HIP) to a neighbor NPU via the mux subsystem (CDCCP protocol handled by a LIP), and to the terminals (various protocols). Standard terminal protocols are handled by the ASYNC, BSC, Mode 4, HASP and X.25 TIPs.

Interrupts -
A set of hardware lines and software programs that allow external events to interrupt NPU processing. Interrupting programs allow preferential processing on a priority basis. The lowest priority level is processed by an OPS monitor.

IVT Commands -
A group of commands that allow the operator at the terminal or a host application program to control some of the IVT transforms made by a TIP. These commands can (1) change the terminal characters for breaks and cancel signs, (2) select output page format (such as page width and length, number of padding characters after a line feed or carriage return), (3) designate parity type, terminal class, and other device-related features.

Level -
For logical records, an octal number 0 through 17 in the system-supplied 48-bit marker that terminates a short or zero-length PRU.

Line -
A connection between an NPU and a terminal, or a group of terminals.

Line Control Block (LCB) -
A table assigned to each active line in the system. It contains configuration information as well as current processing information.

Link -
A connection between two NPUs or an NPU and a host.

Link Interface Package (LIP) -
The CCP program which handles frame transfers across a trunk; that is, across the connection between a local and a remote NPU. A LIP uses CDCCP protocol and interfaces on the local NPU side to the HIP. On the remote NPU side, the LIP interfaces with the appropriate TIP. In both local and remote NPUs, the LIP interfaces with the mux subsystem for transfer across the trunk.

List -
A group of logical connections with the same application list number, which are linked together by NAM and treated as a single entity in NETGETL or NETGTFL calls.

List Number -
See Application List Number.

Load -
The process of moving programs downline from the host and storing them in the NPU main and micromemory. Loading of a remote NPU is accomplished by the host through the use of the LIP in the local NPU.

Local Configuration File (LCF) -
A file in the host computer system, containing information on the physical and logical makeup of the communication elements in the system. The file contains a list of the application programs available for execution in the host computer, and the lines and terminals that can access it. This is a NOS direct access permanent file.

Local NPU -
An NPU which is connected to the host via a coupler. A local NPU always contains a HIP for processing block protocol transfers across the host/local NPU interface. Synonymous with front-end NPU. Contrast with remote NPU.

Logical Channel Control Block (LCCB) -
A data structure holding information about logical channels. These are used by the X.25 TIP for terminals connected to the NOS network through a public data network.

Logical Connection -
A logical message path established between two application programs or between a network terminal and an application program. Until terminated, the logical connection allows messages to pass between the two entities.

Logical Line -
The basic message unit of a device. An upline logical line is designated by a carriage return for upline. See Physical Line.

Logical Link (LL) -
The portion of a logical connection defined by host node and terminal node ID numbers. A logical link is an error-free path across the network over which many separate logical connections are multiplexed. A logical link cannot traverse more than two NPUs.

Logical Link Control Block (LLCB) -
A table assigned to each logical link in the system which touches this NPU. The table contains configuration information as well as current processing information.

Logical Record -
Under NOS, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. Equivalent to a system-logical-record under NOS/BE.

Loop Multiplexer (LM) -
The hardware which interfaces the CLAs (which convert data between bit-serial digital and bit-parallel digital character format) and the input and output loops.

Low/Medimum-Speed Voice-Grade Line -
A line that operates at bit transmission rates at or below 19200 b/s. These lines characteristically connect individual terminals to an NPU or to a PAD access device.

Macromemory -
The portion of 255x Series network processing unit memory that contains code involved in data communication, such as the Terminal Interface Program.

Main Memory -
The macromemory of the NPU. It is partly dedicated to programs and common areas; the remainder is buffer area used for data and overlay programs. Word size is 16 data bits plus three additional bits for parity and program protection. Memory is packaged in 16K and 32K word increments.

Mask -
In CCP, a bit pattern used in the interrupt subsystem to check if an interrupt is of sufficiently high priority to be processed now. An interrupt mask register is used in this processing.

Message -
A logical unit of information, as processed by an application program. When transmitted over a network, a message can consist of one or more blocks.

Micromemory -
The micro portion of the NPU memory. This consists of 4096 words of 16-bit length. 1024 words are Read Only Memory (ROM); the remaining words are Random Access Memory (RAM) and are alterable. The ROM memory contains the emulator microprogram that allows use of assembly language.

Microprocessor -
The portion of the NPU that processes the programs.

Mode 4 -
A communication line transmission protocol which requires the polling of sources for input to the data communication network. Control Data defines two types of mode 4 equipment, mode 4A and mode 4C. Mode 4A equipment is polled through the hardware address of the console device, regardless of how many devices interface to the network. Mode 4C equipment is polled through separate hardware addresses, depending on the point each device uses to interface with the network.

Modem -
A hardware device for converting analog levels to digital signals and the converse. Long lines interface to digital equipment via modems. Modem is synonymous with data set.

Module -
See program.

Monitor -
The portion of the NPU base system software responsible for time and space allocation within the computer. The principal monitor program is OPSMON, which executes OPS level programs by scanning a table of programs which have pending tasks.

Multiplex Loop Interface Adapter (MLIA) -
The hardware portion of the multiplex subsystem that controls the multiplexing loops (input and output) as well as the interface between the NPU and the multiplexing subsystem.

Multiplex Subsystem -
The portion of the base NPU software which performs multiplexing tasks for upline and downline data, and also demultiplexes upline data from the CIB and places the data in line-oriented input data buffers.

Neighbor NPUs -
Two NPUs connected to one another by means of a trunk. The NPU connected to the host via a coupler is designated as the local NPU. The other NPU is a remote NPU; it is not connected directly to the host in any fashion.

Network -
An interconnected set of network processing units, hosts, and terminals.

Network Access Method (NAM) -
A software package that provides a generalized method of using a communication network for switching, buffering, queuing, and transmitting data. NAM is a set of interface routines used by a terminal servicing facility for shared access to a network of terminals and other application programs, so that the facility program does not need to support the physical structures and protocols of a private communication network.

Network Address -
The address used by block protocol to establish routing for the message. It consists of three parts; DN - the destination node, SN - the source node, and CN - the connection number.

Network Configuration File (NCF) -
A network definition file in the host computer, containing information on the network elements and permissible linkages between them. The status of the elements described in this file is modified by the network operator in the course of managing the network through the Network Supervisor. This is a NOS direct access permanent file.

Network Definition File -
Either of the two types of NDL program output files that determine the configuration of the network. This can be a network configuration file or a local configuration file.

Network Definition Language (NDL) -
The compiler-level language used to define the network configuration file and local configuration file contents.

Network Definition Language Processor (NDLP) -
The network software module that processes an NDL program as an off-line batch job to create the network definition files and other NDL program output.

Network Element -
Any configurable entity supervised or loaded by the Network Supervisor. A network element consists of any entity in the total network that is not a communication element; this term is usually applied to the data communication network entities comprising the NPUs and their linkages.

Network Logical Address -
See Network Address.

Network Processing Unit (NPU) -
The collection of hardware and software that switches, buffers, and transmits data between terminals and host computers.

Network Supervisor (NS) -
A portion of the network software, written as a NAM application program. The Network Supervisor dumps and loads the NPUs in the communication network.

Node -
A hardware or software entity that creates, absorbs, switches, and/or buffers message blocks. NPUs and host couplers are communication nodes of the network.

NPU Operator (NOP) -
The operator who resides at a terminal and controls network elements such as NPUs, trunks, logical links, lines, and terminals. Contrast with host operator.

Off-Line Diagnostics -
Optional diagnostics for the NPU that require the NPU be disconnected from the network.

On-Line Diagnostics -
Optional diagnostics for the NPU that can be executed while the NPU is connected to, and operating as a part of the network. Individual lines being tested must, however, be disconnected from the network. These diagnostics are provided if the user purchases a network maintenance contract.

OPS Monitor -
The NPU monitor. See Monitor.

Output -
Information flowing downline from host to terminal.

Output Buffer -
Any buffer that is used to hold a downline message from the host.

Packet -
A group of binary digits, including data and call control signals, which is switched as a single unit. The data, control signals, and error-control information are arranged in a specific format.

Packet Assembly/Disassembly Access (PAD) -
A definition of the procedures for the operation of an asynchronous terminal through a packet-switching network (PSN).

Assembly: The accumulation of characters from an asynchronous device into data blocks for transmission via a PDN. Disassembly: The encoding of blocks for transmission to an asynchronous terminal.

Packet-Switching Network (PSN) -
A packet-switching network provides data communication service between various terminal and computer systems or networks. The PSN is usually licensed as a common carrier.

Terminal interface to a PSN is defined by the packet assembly/disassembly (PAD) access. PSN interface with a NOS network is defined by the X.25 protocol.

PAD SubTIP -
A subTIP of the X.25 TIP that allows asynchronous ASCII terminals to communicate over a public data network.

Paging (NPU) -
A method of executing programs and accessing data in the NPU main memory region above 65K. Paging is required for addressing where the address is larger than 16 bits (NPU word size) in length.

Paging (Screen) -
The process of filling a CRT display with data and holding additional data for subsequent displays. Changing the paged display is terminal operator controlled if the page wait option is selected.

Parameterization -
The process whereby all of the configurable characteristics of a specific model of terminal are reconciled with the characteristics of that terminal's general terminal class. All characteristics not specifically declared for a given terminal are inferred from the terminal's assigned terminal class. Characteristics can be declared through the terminal's definition in the local configuration file, or by the terminal user through dialog with the Terminal Interface Program, or by an application program servicing the terminal.

Parity -
A type of data assurance. The most common parity is character parity; that is, the supplying of one extra bit per character so that the sum of all the bits in the character (including the parity bit) is always an even (even parity) or odd (odd parity) number.

PASCAL -
A high level programming language used for CCP programs. Almost all CCP programs are written in PASCAL language.

Passive Terminal -
Any terminal incapable of conducting both input and output and therefore incapable of dialog with the Network Validation Facility. Batch unit record peripherals are typical examples of passive terminals. Also known as a nonconsole device. Contrast with Interactive Terminal.

Password -
A parameter in the terminal operator's login procedure type-in, used for additional access security by the Network Validation Facility. This parameter does not appear in any supervisory messages.

Peripheral Processor Unit (PPU) -
The hardware unit within the host computer that performs physical input and output through the computer's data channels.

Physical Line -
A string of data that is determined by the terminal's physical characteristics (page width or line feed). Contrast with logical line, which is determined by a carriage return or other forwarding signal.

Physical Link -
A connection between two major network nodes such as neighboring nodes. Messages can be transmitted over active physical links.

Physical Record Unit (PRU) -
Under NOS, the amount of information transmitted by a single physical operation of a specified device. The size of a PRU depends on the device, as shown in table C-2.

A PRU that is not full of user data is called a short PRU; a PRU that has a level terminator but no user data is called a zero-length PRU.

TABLE C-2. PRU SIZE

Device	Size in Number of 60-Bit Words
Mass storage	64
Tape in SI format with binary data	512
Tape in I format	512
Tape in other format	Undefined

Polling -
The process of requesting input from hardware or software that only provides input on request. Polling is a concept of several network protocols and is used to avoid input contention. Mode 4 terminals are polled for input by the Terminal Interface Program servicing them; an application program polls all logical connections for input, whether the logical connections are with controlled mode 4 terminals or free-wheeling asynchronous terminals.

Port (P) -
The physical connection in the NPU through which data is transferred to/from the NPU. Each port is numbered and supports a single line. Subports are possible but not used in this version of CCP.

Primary Function Code (PFC) -
See Function Codes.

Priority -
The condition when traffic through the network is maintained preferentially for one or more terminals out of all terminals producing network traffic. Terminals with priority are the last terminals for which network traffic is suspended when traffic must be temporarily stopped because the network is operating at capacity. Terminals with priority receive preferential treatment of their input or output.

Priority Level -
A set of 17 levels of processing in the NPU. Priority levels are interrupt driven. The OPS monitor processes at the lowest priority level; that is, at a level below any interrupt-driven level.

Program -
A series of instructions which are executed by a computer to perform a task; usually synonymous with a module. A program can be composed of several subprograms.

Protect System -
In the NPU, a method of prohibiting one set of programs (unprotected) from accessing another set of programs (protected) and their associated data. The system uses a protect bit in the main memory word.

Protocol -
A set of standardized conventions which must be used to achieve complete communication between elements in a network. A protocol can be a set of predefined coding sequences, such as the control byte envelopes added to or removed from data exchanged with a terminal; a set of data addressing and division methods, such as the block mechanism used between an application program and the Network Access Method; or a set of procedures used to control communication, such as the supervisory message sequences used between an application program and the Network Access Method.

PRU Commands -
A set of commands from the host or a terminal that changes batch device or batch file characteristics, alters batch data stream flow, or transmits accounting data to the host. All batch file and batch device commands can come from the host. A few batch file commands can also come from the terminal. Some batch stream flow commands come from the host; others come from the terminals. Accounting data commands come only from the terminals.

PRU Device -
Under NOS, a mass storage device or a tape in SI or I format, so called because records on these devices are written in PRUs.

PRUB -
Physical record unit block. A block format for batch terminals that is compatible with the host's PRU (batch file) handling capabilities. CCP TIPs convert all upline batch messages to this format (exception: no transformations are made to transparent data except to put the messages into PRUBs). By this method, application programs in the host need only to be able to process batch data in PRU format rather than in the multiplicity of formats which real terminals use. Downline messages from the host to real batch devices are converted from PRUB to real terminal format. PRUB processing is controlled by the TIPs with the help of the BIP.

Public Data Network (PDN) -
A network that supports the interface described in the CCITT protocol X.25.

Queues -
Sequences of blocks, tables, messages, etc. Most network queues are maintained by leaving the queued elements in place and using tables of pointers to the next queued element. Most queues operate on a first-in-first-out basis. A series of worklist entries for a TIP is an example of an NPU queue.

Random File -
In the context of the NOS operating system, a file with the random bit set in the file environment table in which individual records are accessed by their relative PRU numbers.

Record -
(1) A data unit defined for the host record manager (PRU); (2) a data unit defined for HASP workstations. In either case, a record contains space for at least one character of data and normally has a header associated with it. HASP records can be composed of subrecords.

Regulation -
The process of making an NPU or a host progressively less available to accept various classes of input data. The host has one regulation scheme, the host and multiplex interfaces of a local NPU have another scheme, and the mux interface to a neighbor NPU has a third regulation scheme. Some types of terminals (for instance, HASP workstations) may also regulate data. Messages are classified as supervisory or service (highest priority) priority data and nonpriority data. Priority of data is established on a terminal-to-terminal basis through the PRI classification in NDL.

Remote NPU -
A network processing unit linked indirectly to a host computer through other network processing units. Contrast with local NPU.

Response Messages -
A subclass of supervisory or service messages that is a response to a supervisory or service message of the originator. Response messages normally contain the requested information or indicate that the requested task has been started or performed. Error or abnormal responses are sent when the responder cannot deliver the information or start the task.

Return Parameter -
A parameter in an AIP call that provides as input to the AIP routine the identification of a location to which AIP should transfer information. This location is within the application program's field length and outside of the AIP portion of that field length. A return parameter cannot be a constant or a value in itself. Return parameters are always symbolic addresses. The time at which transfer of information from AIP occurs depends on whether the program is operating in parallel mode and whether use of the parameter is global to all AIP routines or local to the call in which it is used.

Routing -
The process of sending data/commands through the network to its destination (for instance, a terminal). The network logical address (DN, SN, CN) is the primary criterion for routing. In the NPU, directories are used to accomplish the routing function.

Sequential -
A file organization in which records are stored in the order in which they are generated.

Service Channel -
The network logical connection used for service message transmission. For this channel, CN=0. The channel is always configured, even at load time.

Service Message (SM) -
The network method of transmitting most command and status information to/from the NPU. Service messages use CMD blocks in the block protocol.

Service Module (SVM) -
The set of NPU programs responsible for processing service messages. SVM is a part of the BIP.

Short PRU -
A PRU that does not contain as much user data as the PRU can hold, and is terminated by a system terminator with a level number. Under NOS, a short PRU defines EOR.

Source -
The terminal or host computer program that creates a message.

Source Node (SN) -
The node that interfaces directly to the source of a data message block.

State Program Tables -
CCP tables used by the multiplex subsystem to locate the next state program to execute.

State Programs -
CCP programs written in state programming language. These programs usually are part of a TIP (some are common to all TIPs), but do not operate on the OPS level. Instead they are reached by a call to the mux level or are operated automatically by the multiplex subsystem. State programs process modem signals, input data, and output data. Text processing is primarily performed by state programs.

Station -
A provider and/or recipient of data messages; usually synonymous with a grouping of terminals. This term is used in various ways within mode 4 communications documentation, as shown in table C-1.

Statistics Service Message -
A subclass of service messages that contain detailed information about the characteristics and history of a network element such as a line or a terminal.

Status -
Information relating to the current state of a device, line, etc. Service messages are the principal carriers of status information. Statistics are a special subclass of status.

String -
A unit of information transmission used by the HASP protocol. One or more strings compose a record. A string can be composed of different characters or contiguous identical characters. In the latter case, the string is normally compressed to a single character and a value indicating the number of times the character occurs.

Subfunction Code (SFC) -
See Function Codes.

Subport -
One of several addresses in a port. In this release of CCP, subport is always equal to 0.

Subprogram -
A series of instructions which are executed by a computer to perform a task or part of a task. A subprogram may be called by several programs or may be unique to a single program. Subprograms are normally reached by a direct call from a program.

Supervisory Message -
A message block in the host not directly involved with the transmission of data, but which provides information for establishing and maintaining an environment for the communication of data, between the application program and NAM, and through the network to a destination or from a source. Supervisory messages may be transmitted to an NPU in the format of a service message.

Switched Line -
A communication line connected with one network processing unit but able to be connected to any one of several terminals via a switching mechanism, such as a dialed telephone line.

Switching -
The process of routing a message or block to the specified internal program or external destination.

Symbolic Address -
The abstract identification of an entity serving as a location from which or to which information can be transferred. A symbolic address can contain information, but does not constitute information. A symbolic address is an identifier represented in character form by

the programmer, and is equivalent to the concept of a variable in the terminology of some programming languages. In FORTRAN or ALGOL programs, typical symbolic addresses include array names, array element names, and variable names. In COMPASS, a symbolic address is equivalent to a label in a source code location field; a relative address cannot be used as a symbolic address. In COBOL, a symbolic address is equivalent to a level 01 Data Description entry. In SYMPL, a symbolic address is equivalent to the name of an array or scalar item in a data declaration.

Synchronous -
A transmission in which data is transferred in blocks that are framed by start and end-of-text characters. The blocks of data can be comprised of characters of any predefined bit length.

Character synchronization is achieved by recognition of a predefined sync character that precedes the block of data.

System Configuration -
The process of setting tables and variables throughout the network to assign lines, links, terminals, etc., so that all elements of the network recognize a uniform addressing scheme. After configuration, network elements accept all data commands directed to/through themselves and reject all other data and commands.

Terminal -
An entity, external to the data communication network but connected to it via a communication line, that supplies input messages to, and/or accepts output messages from, an application program. In the context of this manual, a terminal is each separately addressable device comprising a physical terminal or station.

Terminal Address -
The hardware address of a mode 4 console or mode 4C printer. This term is used in various ways within mode 4 communications documentation, as shown in table C-1.

Terminal Class (TC) -
An NDL parameter and supervisory message field value describing the physical attributes of a group of similar terminals, in terms of an archetype terminal for the group.

Terminal Control Block (TCB) -
A control block within CCP containing configuration and status information for an active terminal. TCBs are dynamically assigned.

Terminal Interface Program (TIP) -
A portion of the Communications Control Program that provides an interface for terminals connected to a 255x Series network processing unit. The TIP performs character conversion to and from 8-bit ASCII, limited editing of the input and output stream, parity checking, and so forth.

Terminal Name (TNAME) -
A name of up to seven letters and digits known to the network and used to identify a terminal to the network's local operator.

Terminal Node -
The node number associated with an NPU that interfaces with a terminal.

Terminal Operator -
The person operating the controls of a terminal. Contrast with user.

Terminal Servicing Facility -
See Application Program.

Test Utility Program (TUP) -
A debugging utility that supports breakpoint debugging of CCP as well as other utility type operations such as loading and dumping.

Text Area (TA) -
The area within the application program that receives the message block text from a NETGET, NETGETF, NETGTFL, or NETGETL call, or contains the message block text for a NETPUT or NETPUTF call.

Text Length in Characters (TLC) -
A field in the application block header specifying the number of character bytes of text in the message block.

Text Length Maximum (TLMAX) -
Maximum length in central memory words of the data message text block that the application program will accept for processing.

Timeout -
The process of setting a time for completion of an operation and entering an error processing condition if the operation has not finished in the alloted time.

Timing Services -
The subset of base system programs within CCP which provide timeout processing and clock times for messages, status, etc. Timing services provide the drivers for the real-time clock.

Trailer -
Control information appended to the end of a message unit. A trailer contains the end-of-data control signals. Trailers can be generated by the terminal or by an intermediate device such as a frame generator. Not all headers are matched with trailers, although some devices split their control information between a header and a trailer. The trailer usually contains a data assurance field such as a CRC-16 or a checksum. Like headers, trailers are generated and discarded at various stages along a message unit's path.

Transparent Mode -
A software feature provided by the Network Access Method and the network processing unit TIP. When transparent mode transmission occurs between an application program and a terminal, the Network Access Method does not convert data to or from display code, and the TIP does not edit the character stream or convert the characters to or from 8-bit ASCII code. When no parity is in effect for the terminal and transparent mode transmission occurs, all eight bits of the character byte can be used to represent characters in 256-character sets (such as EBCDIC).

Trunk -
The dedicated communication line connecting two network processing units.

Trunk Protocol -
The protocol used for communicating between neighboring NPUs. It is modified CDCCP protocol which uses the frame as the basic communications element.

Typeahead (Terminal) -
The ability of a terminal to enter input data at all times without losing output data currently in progress. This requires suspending the output operation until the input message is finished, and then resuming the interrupted output. The ASYNC TIP supports typeahead; the X.25 TIP supports typeahead if it is provided by the PDN.

Unsolicited Service Message -
Service messages sent to the host which do not respond to a previous service message from the host. Unsolicited SMs report hardware failures to the host.

Upline -
The direction of input flow from terminal through an NPU to host computer.

User -
That person or group of people who are the preparers and/or recipients of messages communicated with an application program via the network. A user may interface with one or more terminals, or with no terminals. Contrast with terminal operator.

User Name -
The NOS validation file parameter entered by the terminal operator during the Network Validation Facility log-in procedure.

Virtual Channel (X.25/PAD) -
A channel defined for moving data between a terminal and a host. Virtual channels are defined for the length of time that the terminal is connected to the PDN.

Word -
The basic storage and processing element of a computer. The NPU uses 16-bit word (main memory) and 32-bit word (internal to the microprocessor only). All interfaces are 16-bit word (DMA) or in character format (multiplex loop interface). Characters are stored in main memory two per word. Hosts (CYBER series) use 60-bit words but a 12-bit byte interface to the NPU. Characters at the host side of the NPU host interface are stored in bits 19 through 12 and 7 through 0 of a dual-12-bit type.

Some terminals such as a HASP workstation can use any word size but must communicate to the NPU in character format. Therefore, workstation word size is transparent to the NPU.

Worklist Processor -
Within CCP, the base system programs responsible for creating and queuing worklist entries.

Worklists -
Within CCP, packets of information containing the parameters for a task to be performed. Programs use worklists to request tasks of OPS level programs. Worklist entries are queued to the called program. Entries are one to six words long, and a given program always has entries of the same size.

X.25 Protocol -
A CCITT protocol used by the public data network. It is characterized by high-speed, framed data transfers over links. A PDN requires a PAD access for attaching asynchronous terminals.

X.25 TIP -
The CCP TIP that interfaces an NPU to a public data network.

Zero-Length PRU -
A PRU that contains system information, but no user data. Under NOS, a zero-length PRU defines EOF.

MNEMONICS

Following is a list of mnemonics used in this manual:

ABH	Application Block Header
ABL	Application Block Limit
ABN	Application Block Number
ABT	Application Block Type
ACK	Positive Acknowledgment Block
ACK0/ACK1	Acknowledge Block (BSC/HASP protocol)
ACN	Application Connection Number
ACT	Application Character Type
ACTL	Assurance Control Block
AIP	Application Interface Program
ALN	Application List Number
ANAME	Application Name
APL	A Programming Language
APP	Application
ARM	Asynchronous Response Mode
ASCII	American Standard Code for Information Interchange
ASYNC	Asynchronous
BCB	Block Control Byte (HASP protocol)
BCD	Binary Coded Decimal
BFC	Block Flow Control
BFR	Buffer
BIP	Block Interface Package
BLK	Message Block
BN	Block Number
BRK	Break Block
BSC	Binary Synchronous Communication
BSN	Block Serial Number
BSZ	Block Size
BT	Block Type
B1, B2	User-defined breaks
CA	Cluster Address
CB	Control Block
CCITT	Comite Consultif International Telephonique et Telegraphique (an international communications standards organization)
CCP	Communications Control Program
CDCCP	CDC Communications Control Procedure
CDT	Conversational Display Terminal
CE	Customer Engineer
CFS	Configurator State (for SVM)
CIB	Circular Input Buffer
CLA	Communications Line Adapter
CMD	Command Block
CMDR	Command Reject (trunk protocol)
CN	Connection Number (for blocks/SVM)
CND	Connection Number Directory
CR	Carriage Return
CRC	Cyclic Redundancy Check
CRT	Cathode Ray Tube
CS	Communications Supervisor
CTL	Control Element (ASYNC protocol)
DBC	Data Block Clarifier (for blocks/SVM)
DBZ	Downline Block Size
DCB	Diagnostics Control Block
DEL	Delete Character
DLFP	Debug Log File Postprocessor utility
DM	Disconnect Mode (trunk protocol)
DMA	Direct Memory Access (in NPU)
DN	Destination Node (for blocks/SVM)
DND	Destination Node Directory
DSR	Data Set Ready
DT	Device Type
EBCDIC	Extended Binary Coded Decimal Interchange Code
EC	Error Code
E-CODE	Device Codes (Mode 4 protocol)
ENQ	Enquiry Block (BSC/HASP protocol)
EOF	End of File
EOI	End of Information
EOJ	End of Job
EOM	End of Message
EOR	End of Record (HASP protocol)
EOT	End of Transmission

ETB	End of Transmission Block (HASP protocol)
ETX	End of Text
FCD	First Character Displacement (in buffer)
FCS	Function Control Sequence (HASP protocol)
FD	Forward Data (block protocol)
FDX	Full Duplex
FE	Format Effector
FET	File Environment Table
FF	Forms Feed
FN	Field Number (for SVM)
FRQ	Frame Retention Queue (trunk protocol)
FS	Forward Supervision (block protocol)
FV	Field Values (for SVM protocol)
HA	Header Area
HASP	Houston Automatic Spooling Protocol
HCP	Host Communications Processor (alternate name for NPU)
HDLC	High Level Data Link Control
HDX	Half Duplex
HIP	Host Interface Package
HL	High Level
HN	Host Node
HO	Host Ordinal
HOP	Host Operator
IAF	Interactive Facility program
ICMD	Interrupt Command
ICMDR	Interrupt Command Response
ICT	Input Character Type
ID	Identifier (number of code)
IDC	Internal Data Channel (in NPU)
IDP	Input data processor
I-FRAME	Information Frame (for trunk protocol)
INITN	Initialization Block Acknowledgement
INITR	Initialization Block Request
I/O	Input/Output
ISO	International Standards Organization
IVT	Interactive Virtual Terminal Format
LBN	Last Block Number
LCB	Line Control Block
LCCB	Logical Channel Control Block
LCD	Last Character Displacement
LCF	Local Configuration File
LD	Load/Dump
LF	Line Feed
LFG	Load File Generator
LIDLE	Idle Element (trunk protocol)
LINIT	Line Initialization Element (trunk protocol)
LIP	Link Interface Package
LISTPPM	PID Dump Analyzer
LL	Logical Link
LLCB	Logical Link Control Block
LLREG	Logical Link Regulation
LM	Loop Multiplexer
LP	Line printer
LRN	Link Remote Node (for SVM)
LT	Line Type
MCS	Message Control System
MLCB	Multiplex Line Control Block
MLIA	Multiplex Loop Interface Adapter
MM	Main Memory
MPLINK	The PASCAL link editor
MSG	End-of-Message Block
MTI	Message Type Indicators (Mode 4 protocol)
M4	Mode 4
NAK	Negative Acknowledgment Block (BSC/HASP protocol)
NAM	Network Access Method
NCB	Network Configuration Block
NCF	Network Configuration File
NDA	Network Dump Analyzer
NDLP	Network Definition Language Processor
NIP	Network Interface Program

NLF	Network Load File
NOP	NPU Operator
NPU	Network Processing Unit
NS	Network Supervisor program
NVF	Network Validation Facility
ODD	Output Data Demand (Multiplex subsystem)
ODP	Output Data Processor
OPS	Operational (OPS level = Monitor level programs)
OPSMON	Monitor
P	Port
PAD	Packet Assembly/Disassembly
PDN	Public Data Network
PFC	Primary Function Code (for SVM)
PIP	Peripheral Interface Program
PL	Page Length (IVT)
PPU	Peripheral Processing Unit
PRU	Physical Record Unit
PRUB	Physical Record Unit Block
PSN	Packet Switching Network
PW	Page Width
QDEBUG	PASCAL Debugging Package
QTRM	Queued Terminal Record Manager
RAM	Random Access Memory
RBF	Remote Batch Facility program
RC	Reason Code (for SVM) (also called response code)
RCB	Record Control Byte (HASP protocol)
RCV	Receive State
REJ	Reject (trunk or X.25 protocol)
RIM	Request Initialization Mode (trunk protocol)
RL	Regulation Level
RM	Response Message (SM)
RNR	Receive Not Ready (trunk or X.25 protocol)
ROM	Read Only Memory
RR	Receive Ready (trunk or X.25 protocol)
RS	Reverse Supervision (block protocol)
RST	Reset Block
RT	Record Type
RTS	Request to Send (trunk protocol)
SAM-P	System Autostart Module Program
SARM	Set Asynchronous Mode (trunk or X.25 protocol)
SCB	String Control Byte (HASP protocol)
SFC	Secondary Function Code (for SVM)
S-FRAME	Supervisory Frame (trunk or X.25 protocol)
SIM	Set Initialization Mode (trunk protocol)
SM	Service Message
SN	Source Node (for blocks/SVM)
SND	Source Node Directory
SP	Subport
SRCB	Subrecord Control Byte (HASP protocol)
STX	Start of Text (ASYNC protocol)
SVM	Service Module (for processing service messages)
SYNC	Synchronizing Element (Mode 4 protocol)
TA	Terminal Address
TAA	Text Area Array
TAF	Transaction Facility
TC	Terminal Class
TCB	Terminal Control Block
TDP	Time Dependent Program
TERM	Termination Block
TIP	Terminal Interface Program
TIPTQ	TIP Trunk Queues (trunk protocol)
TLC	Text Length in Characters
TLMAX	Text Length Maximum
TNAME	Terminal Name
TO	Timeout
TOT	Total Number of Trunks (SM)
TPCB	Text Processor Control Block
TT	Terminal Type
TTF	Trunk Transmission Frame
TTY	Teletype writer

TUP	Test Utility Program
TVF	Terminal Verification Facility
UA	Unnumbered Acknowledgment (trunk or X.25 protocol)
UBZ	Upline Block Size
U-FRAME	Unnumbered Frame (see UA and UI)
UI	Unnumbered Information frame (trunk or X.25 protocol)
US	Unit Separator
UT	User Terminal
VAR	PASCAL Variable
WL	Worklist
WLCB	Worklist Control Block
WLE	Worklist Entry
WLP	Worklist Processor
XBZ	Transmission Block Size
X-OFF	Stop character (ASYNC protocol)
X-ON	Start character (ASYNC protocol)
XPT	Transparent bit
X.3	CCITT protocol for asynchronous terminal access to a PDN
X.25	CCITT protocol for public data network
X.28	CCITT protocol for terminal access to PDN/PAD
X.29	CCITT protocol for host access to PDN/PAD

USER PROGRAM CALL STATEMENT SUMMARY D

This appendix summarizes the formats of calls to AIP and QTRM routines. The general format of each routine is listed alphabetically without description opposite the page number where the routine is completely described.

COMPILER LEVEL (NETIO-RESIDENT OR NETIOD-RESIDENT)

Call Format	Page
CALL NETCHEK	5-19
CALL NETDBG(dbugsup,dbugdat,avail)	6-5
CALL NETDMB(dumpid,ecs)	6-7
CALL NETGET(acn,ha,ta,tlmax)	5-4
CALL NETGETF(acn,ha,na,taa)	5-6
CALL NETGETL(aln,ha,ta,tlmax)	5-11
CALL NETGTFL(aln,ha,na,taa)	5-13
CALL NETLGS(address,size)	6-13
CALL NETLOG(address,size,format)	6-7
CALL NETOFF	5-3
CALL NETON(aname,nsup,status,minacn, maxacn)	5-1
CALL NETPUT(ha,ta)	5-8
CALL NETPUTF(ha,na,taa)	5-9
CALL NETREL(lfn,msglth,nrewind)	6-6
CALL NETSETF(flush,fetadr)	6-6
CALL NETSETP(option)	5-17
CALL NETSTC(onoff,avail)	6-13
CALL NETWAIT(time,flag)	5-15
CALL NSTORE(array,field,value)	4-12
[ivalue=]NFETCH(array,field)	4-12
ENTER FORTRAN-X QTCLOSE	E-14
ENTER FORTRAN-X QTENDT	E-14
ENTER FORTRAN-X QTGET USING ta-in-acn$_i$	E-12
ENTER FORTRAN-X QTLINK	E-13
ENTER FORTRAN-X QTOPEN USING net-info-table	E-10
ENTER FORTRAN-X QTPUT USING ta-out-acn$_i$	E-11
ENTER FORTRAN-X QTTIP USING ta-out-acn$_i$	E-15

ASSEMBLY LANGUAGE LEVEL (NETTEXT-RESIDENT)

Call Format		Page
[label] NETCHEK		5-19
[label] NETDBG	dbugsup,dbugdat, avail	6-5
label NETDBG	dbugsup,dbugdat, avail,LIST	6-5
[label] NETDBG	{LIST=label / LIST=register name}	6-5
[label] NETDMB	dumpid,ecs	6-7
label NETDMB	dumpid,ecs,list	6-7
[label] NETDMB	{LIST=label / LIST=register name}	6-7
[label] NETGET	acn,ha,ta,tlmax	5-4
label NETGET	acn,ha,ta,tlmax, LIST	5-4
[label] NETGET	{LIST=label / LIST=register name}	5-4
[label] NETGETF	acn,ha,na,taa	5-6
label NETGETF	acn,ha,na,taa,LIST	5-6
[label] NETGETF	{LIST=label / LIST=register name}	5-6
[label] NETGETL	aln,ha,ta,tlmax	5-11
label NETGETL	aln,ha,ta,tlmax,LIST	5-11
[label] NETGETL	{LIST=label / LIST=register name}	5-11
[label] NETGTFL	aln,ha,na,taa	5-13
label NETGTFL	aln,ha,na,taa,LIST	5-13

Call Format		Page
[label] NETGTFL	{LIST=label / LIST=register name}	5-13
[label] NETLGS	address,size	6-13
label NETLGS	address,size,LIST	6-13
[label] NETLGS	{LIST=label / LIST=register name}	6-13
[label] NETLOG	address,size,format	6-7
label NETLOG	address,size,format, LIST	6-7
[label] NETLOG	{LIST=label / LIST=register name}	6-7
[label] NETOFF		5-3
[label] NETON	aname,nsup,status, minacn,maxacn	5-1
label NETON	aname,nsup,status, minacn,maxacn,LIST	5-1
[label] NETON	{LIST=label / LIST=register name}	5-1
[label] NETPUT	ha,ta	5-8
label NETPUT	ha,ta,LIST	5-8
[label] NETPUT	{LIST=label / LIST=register name}	5-8
[label] NETPUTF	ha,na,taa	5-9
label NETPUTF	ha,na,taa,LIST	5-9
[label] NETPUTF	{LIST=label / LIST=register name}	5-9
[label] NETREL	lfn,msglth,nrewind	6-6
label NETREL	lfn,msglth, nrewind,LIST	6-6
[label] NETREL	{LIST=label / LIST=register name}	6-6
[label] NETSETF	flush,fetadr	6-6
label NETSETF	flush,fetadr,LIST	6-6
[label] NETSETF	{LIST=label / LIST=register name}	6-6
[label] NETSETP	option	5-17
label NETSETP	option,LIST	5-17
[label] NETSETP	{LIST=label / LIST=register name}	5-17
[label] NETSTC	onoff,avail	6-13
label NETSTC	onoff,avail,LIST	6-13
[label] NETSTC	{LIST=label / LIST=register name}	6-13
[label] NETWAIT	time,flag	5-15
label NETWAIT	time,flag,LIST	5-15
[label] NETWAIT	{LIST=label / LIST=register name}	5-15
[label] NFETCH	array,field,{Xj / Bj}	4-10
[label] NSTORE	array,field=value	4-11

QUEUED TERMINAL RECORD MANAGER E

The Queued Terminal Record Manager (QTRM) utility package allows an application program to use NAM to perform input and output to and from a terminal or application in a way similar to the use of the CYBER Record Manager to perform input and output to and from mass storage. This appendix describes the interface between QTRM and an application program.

NAM allows an application program to communicate with another application program the same as the program does with a terminal. The program then has a connection with a terminal or an application. When the term connection is used in this appendix, it refers to the general case and includes both terminal-to-application connections and application-to-application connections.

An application program interface with QTRM has two parts:

- A formal data structure, called the network information table, is used as a communication area.
- A set of subroutines is used by the application program to perform network actions.

NETWORK INFORMATION TABLE

An application program uses the network information table to communicate with QTRM and with the network software through QTRM. The application program creates the network information table within its own field length. If the program uses overlays, the network information table must be created within the main (0, 0 level) overlay. The length of the network information table varies according to the number of connections the application program is written to communicate with.

The network information table has the format shown in figure E-1. This table is defined so that its first word begins at a word boundary. In a FORTRAN program, the table would be created as one or more one-dimensional arrays. In a COBOL program, the table would be created as a Data Division item beginning with an 01 level description, preferably in the Working Storage section.

The network information table has two consecutive parts. The first portion is a 10-word entry global to program use of the network. The second portion consists of 10-word entries unique to each connection serviced by the application program.

The global portion of the network information table contains a few fields that QTRM writes for the application program to read. Most of the fields in this portion are read or written by either QTRM or the application program.

The connection portion of the network information table contains fields written by QTRM that should only be used by the application program as read-only fields. Errors can result if the application program writes in any of these fields.

The first 9 words of each 10-word entry in the second portion of the table are maintained by QTRM for each connection. Both QTRM and the application program access a given 10-word entry using the application connection number assigned by the network to the connection. For example, if a terminal or application is assigned to connection number 3, QTRM writes all information concerning that terminal or application into the third 10-word entry in the connection portion of the network information table. If the application program needs some information concerning the terminal or application assigned to connection number 5, it reads the fifth 10-word entry in the connection portion of the network information table. The connection number assigned to the terminal or application is therefore an indexing integer that can be used to access the correct 10-word entry in the table, or other tables maintained by the application program to contain information related to servicing the same terminal or application.

The tenth word of the global portion and the tenth word of each of the connection entries are not accessed by QTRM. They are reserved for installation use.

The application program determines the number of 10-word entries in the second portion of the network information table. One 10-word entry must exist for each terminal or application the program is written to service simultaneously. The application program places the number of 10-word entries in the first portion of the network information table so that QTRM knows how many entries exist.

The application program does not need to provide a 10-word entry for each terminal or application serviced cumulatively during a single program execution. The network reassigns a connection number when a terminal or application disconnects from the program, so that several terminals or applications can sequentially use the same connection number at different periods during a single program execution. For example, if the program is intended to service eight terminals at the same time, it provides eight 10-word entries. During a single execution, six different terminals might use each of those entries in succession, but each terminal uses only the entry assigned to it while it communicates with the program. Consequently, the program does not need 48 entries to allow for the possibility.

In figure E-1, the number of 10-word entries is shown as n and is communicated to QTRM as the value in the num-conns field. The connection number for a specific terminal or application is identified as i in the field descriptions.

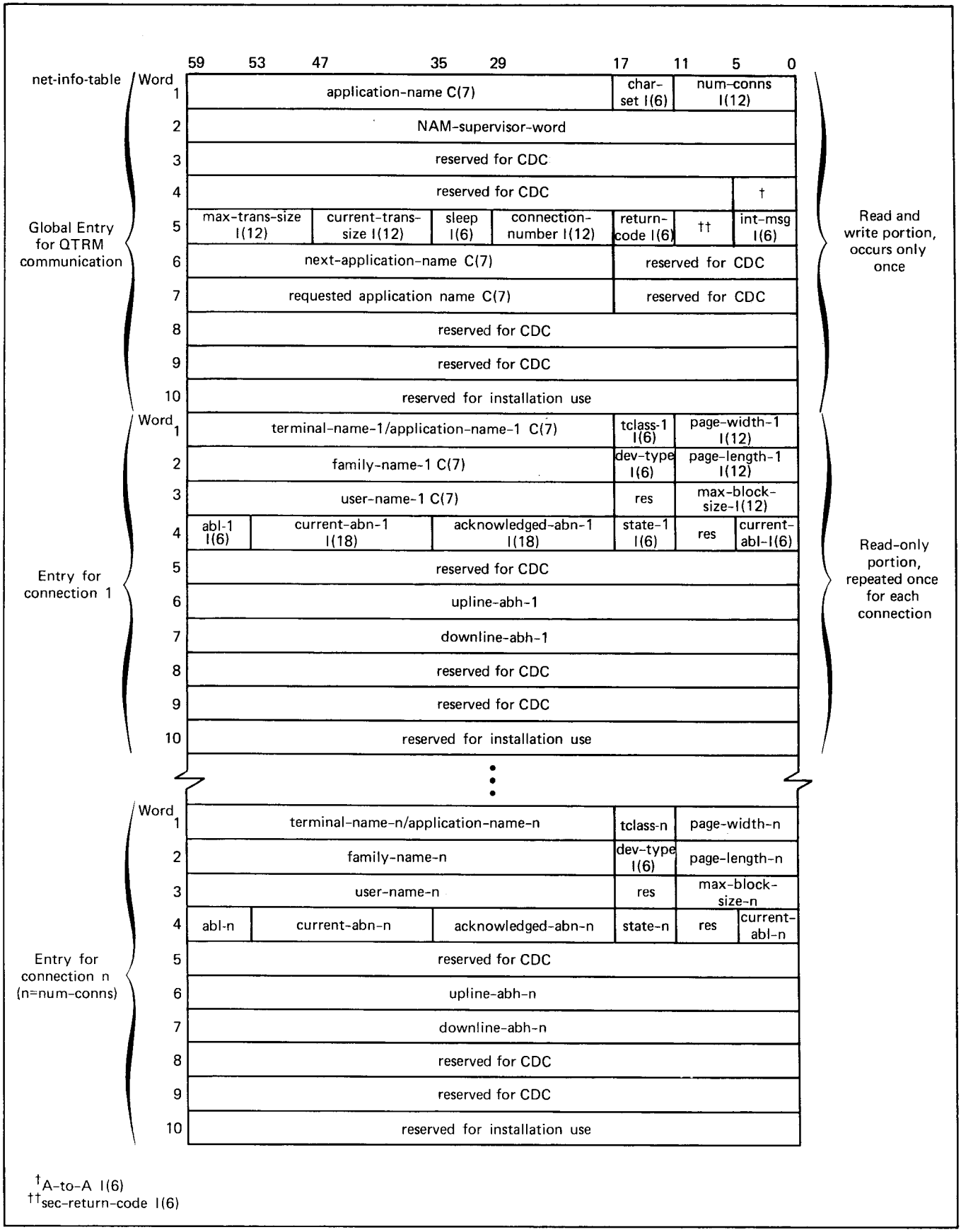


Figure E-1. Network Information Table Format (Sheet 1 of 9)

net-info-table

The symbolic address of the entire network information table, used to identify the table in a QTOPEN call. In a COBOL program, this address is the Data Division descriptor for the level 01 data item containing level 02 or lower level data items for all of the fields described in this figure. In a FORTRAN program, this address is the name of a one-dimensional array.

application-name

This 42-bit field contains the application name used to identify the program to the network, and entered by a terminal user to log in to the program. The name contained in this field can be one to seven letters or digits, beginning with a letter, and must be left-justified within the field and blank-filled to the right; the name must be placed in the field before calling QTOPEN. Changing the contents of this field after calling QTOPEN has no effect. The name placed in this field is subject to the same restraints as the aname parameter in a call to the AIP routine NETON, as described in section 5.

char-set

This 6-bit field contains a binary integer to identify the character code set and byte packing convention along with the mode of data used by the program for all input and output through QTRM.

For input, specify any integer from the following list. Either place the code value in the char-set field before calling QTOPEN, or allow QTOPEN to place the default value of 4 in the char-set field if the application program does not specify a code value.

1 A 60-bit character is in 60-bit word (allowed only for connections to other applications in the same host).

2 8-bit ASCII codes are packed with 7.5 bytes per 60-bit word (every two words contains 15 characters) and transmitted in character mode.

3 8-bit ASCII codes are packed with 5 bytes per 60-bit word (each character code is right-justified within a 12-bit byte and zero-filled to the left) and transmitted in character mode.

4 6-bit display codes are packed with 10 bytes per 60-bit word (this is the default value used by QTRM when no other legal value is specified).

Note that the char-set value at QTOPEN applies to all input from all connections. When a char-set value of 1 is used, only connections to other applications should be made. Char-set values of 2 and 3 can be used for either terminals or applications.

After a call to QTOPEN is made, the char-set field is used to specify a value that applies to output. The application program may change the contents any time. The output is controlled by the char-set value outstanding when QTPUT is called. No QTRM routine changes the contents after QTOPEN is completed. In addition to the code values listed above for input, the following codes are valid for output:

10 8-bit codes are packed with 7.5 bytes per 60-bit word and transmitted in transparent mode.

11 8-bit codes are packed with 5 bytes per 60-bit word and transmitted in transparent mode.

Use of the default value (display code) for output allows use of QTRM editing features. Requirements on the length and contents of the transmitted data are described in section 3 for the act and mode of the application block header.

num-conns

This field contains a 12-bit integer, $1 \leq$ num-conns ≤ 4095, indicating with how many connections the application program can simultaneously support. Connections are assigned numbers from 1 to num-conns; the value used for num-conns should not be greater than the number of 10-word entries provided in the network information table. The network information table must be 10+(10X num-conns) central memory words in length, regardless of whether the program references words at the end of the table. The value must be placed in this field before the call to QTOPEN. After the call to QTOPEN, changing the contents of the field has no effect.

Figure E-1. Network Information Table Format (Sheet 2 of 9)

NAM-supervisor-word	This 60-bit field is used by QTRM and should be ignored by the application program. The field contains the NETON call nsup parameter used by QTRM. (See section 5.)
A-to-A	This 6-bit field contains an integer indicating whether the application program supports application-to-application connections. These application-to-application connections may be initiated by this or another application. This field can contain the following: 0 Does not support application-to-application connections. 1 Supports application-to-application connections. The value must be placed in this field before the QTOPEN. After the call to QTOPEN, changing the contents of the field has no effect.
max-trans-size	This 12-bit field contains a binary integer that indicates the extent of the application program storage area from which data for a connection is sent or into which data is written. The value used is specified in units determined by the code value that is the char-set value at QTOPEN for input and current char-set value for output, as follows: If code = 1, one max-trans-size unit = 60 bits. If code = 2 or 10, one max-trans-size unit = 8 bits. If code = 3 or 11, one max-trans-size unit = 12 bits. If code = 4, one max-trans-size unit = 6 bits. The value used in this field is subject to the following restrictions: Max-trans-size must be less than the number of units that would occupy 410 central memory words. Max-trans-size must be less than 2043 units. Max-trans-size must be at least 11 units longer that the value in the current-trans-size field, if char-set = 4. Max-trans-size must be less than or equal to the number of units that can be contained in the text area (working-storage area) used by the program. Max-trans-size must be set to a value that can be contained exactly in a multiple of central memory words, otherwise QTRM restricts the size of the text area without warning the application to make the last character position end on a word boundary. The value must be placed in this field before any QTPUT or QTGET call, and can be changed between calls as appropriate. This field performs a function comparable to the tlmax parameter in direct AIP routine calls, as described in section 5.
Current-trans-size	This 12-bit field contains a binary integer that indicates how much of the application program text area contains data meaningful for a given QTGET or QTPUT call. The value used is specified in units determined by the code value that is the char-set value at QTOPEN for input and current char-set value for output, as follows: If code = 1, one current-trans-size unit = 60 bits. If code = 2 or 10, one current-trans-size unit = 8 bits. If code = 3 or 11, one current-trans-size unit = 12 bits. If code = 4, one current-trans-size unit = 6 bits.

Figure E-1. Network Information Table Format (Sheet 3 of 9)

On return from a QTGET call that delivers a data block to the program, QTRM places a value in this field that indicates the size of the delivered block. Before a QTPUT call, the application program must set a value in this field that indicates to QTRM the size of the block to be transferred. For char-set values other than 4, the application program must indicate how many units comprise the block (including all ASCII unit separator character codes and any format effector characters). For a char-set value of 4, the application program can use a value of 0, or the nonzero value indicating how many units comprise the block (including all zero byte separators except the last and all format effector characters). Special QTRM output editing functions are performed for data blocks with a char-set of 4, depending on the value in the current-trans-size field; these functions are described in the text under the heading Editing Display-Code Output.

sleep

This 6-bit field contains a signed integer that tells QTRM what action to take after the application program issues a QTGET call. This field can have the values:

- -n Where $1 \leq n < 32$; if no data block or return-code information other than a value of 1 is available to return, the program is suspended by QTRM until information becomes available. If information is available, control returns to the program immediately. The value used for n is not significant.
- 0 The QTGET call is not associated with program suspension; if no data block is available, control returns to the program immediately and a return-code value of 1 is used to indicate the condition to the program. If a block is available, control also returns to the program immediately.
- +n Where $1 \leq n < 32$; the program will be suspended for a maximum of n seconds. Control is returned to the program as soon as any information is available (the return-code field value is not 1) or when the current-abl-i field value is increased for any connection (the return-code field value is 1). If no information is available after n seconds, control is returned to the program with a reason-code field value of 1.

The application program must set or change the value in this field as necessary before each QTGET call. QTRM does not change the value in this field.

connection-number

This 12-bit field contains an integer that identifies the connection involved in the current QTGET, QTPUT, or QTENDT call. On return from a QTGET call, QTRM places the connection number in this field for the connection for which information was returned by the call. Before a QTPUT or QTENDT call, the application program must place the connection number in this field for the connection involved in the call. This value can be used as a subscriptor or index value to access the corresponding 10-word connection entry in the network information table.

return-code

This 6-bit field is used by QTRM to indicate program or connection processing status on return from a QTGET, QTPUT, or QTLINK call. The application program should always test the contents of this field after a QTGET, QTPUT, or QTLINK call. This field can contain the following values:

- 0 Information has been exchanged with the network. After a QTGET, this value indicates that a block was received from a connection and is in the application program text input area identified for that QTGET call; the connection number of the connection generating the block is in the connection-number field of this table. After a QTPUT, this value indicates that the block was given to NAM (however, the block might not have been delivered to the connection yet).

  After a QTLINK call has been made by the program, this value indicates that the request for connection to an application is being forwarded to NAM and is outstanding.
- 1 No information has been exchanged with the network. This value only occurs after a QTGET call that was made while the sleep field contained 0 or a positive value.

Figure E-1. Network Information Table Format (Sheet 4 of 9)

2 A new terminal or application connection has occurred. This value only occurs after a QTGET call. The connection number of the new connection is in the connection-number field, but no data block has been returned by the QTGET call; the 10-word entry in the network information table has been updated by QTRM for the new connection.

3 An improperly formatted block has been detected. This value only occurs as a result of a QTPUT call to a terminal, and usually indicates a missing or misplaced unit separator or zero byte terminator within the block. The block causing the problem and any other subsequent blocks sent to the terminal were discarded by the network.

4 Reserved for CDC use.

5 The current-abl value for the terminal identified in the connection-number field has been exceeded. This reture-code value only occurs after a QTPUT call is attempted when the current-abl value for the terminal is zero. The block involved in the call is discarded by QTRM and must be resent after QTRM resets the current-abl field for the terminal to a nonzero value.

6 The connection between NAM and the terminal or application identified in the connection-number field has been broken by one of the following conditions:

The terminal user hung up.

The communication line failed.

A block sent to the terminal or application was lost by the network.

A block to or from the terminal or application was too long to deliver.

The terminal sent transparent data to the program.

The other application terminated or terminated the connection.

No additional communication is possible between the application program and that terminal or application, and QTENDT should not be called. The information in the 10-word entry for the affected connection remains unchanged until a new connection is made that uses the same entry.

7 The user at the terminal identified in the connection-number field has entered a user-break-1 character. This value only occurs after a QTGET call. On return from the call, QTRM has reset the current-abl field for the affected terminal to the value in the terminal abl field; this change indicates that any blocks previously sent by the program but not yet delivered to the terminal were discarded. The action taken by the application program is determined by what the terminal user expects to occur after entry of the character.

8 The user at the terminal identified in the connection-number field has entered a user-break-2 character. This value only occurs after a QTGET call. On return from the call, QTRM has reset the current-abl field for the affected terminal to the value in the terminal abl field; this change indicates that any blocks previously sent by the program but not yet delivered to the terminal were discarded. The action taken by the application program is determined solely by what the terminal user expects to occur after entry of the character.

9 The network is shutting down. All terminal users should be notified and QTCLOSE should be called as soon as no data blocks are outstanding in either direction.

Figure E-1. Network Information Table Format (Sheet 5 of 9)

10 The network has ended all communication with the application program. This value only occurs after a QTGET call; normally, this value means that the application program should close all files and end its execution. No calls to QTRM routines can be made after receipt of this reason-code value; a call to QTCLOSE is not necessary.

11 The application program has performed some operation that violates NAM protocols. QTRM has received a logical error supervisory message from NAM, as described in section 3. QTRM aborts the program but places the reason code from the supervisory message in the sec-return-code field of the network information table.

12 Another application-to-application request from this program is outstanding. This value is returned by a QTLINK request. The QTLINK request must be reissued after the outstanding request is completed or rejected.

13 The connection was not established. This value is returned by a QTGET call issued by the program following a QTLINK request. The sec-return-code field contains one of the following:

The reason code from the abnormal response to the request-for-connection supervisory message (CON/ACRQ/A) issued by QTRM

The reason code plus 32 from the connection-broken supervisory message (CON/CB/R) if the connection was broken before the connection-processing was completed

14 The application-to-application connection is completed. This value is returned by a QTGET call issued by the program following a QTLINK request. The connection-number field contains the new connection number. The 10-word entry in the network information table has been updated with the new connection information.

15 thru 63 Reserved for CDC use.

sec-return-code — This 6-bit field contains one of the integer logical error supervisory message reason codes described in section 3. This field is not written by the application program, but is provided for debugging.

When the value of the return-code field is set to 11 or 13, this 6-bit field contains additional information for debugging based on reason codes returned in the CON/ACRQ/A and CON/CB/R supervisory messages described in section 3.

int-msg — This 6-bit field contains an integer that indicates to QTRM whether the block involved in a QTPUT call is or is not the last or only block of a message. If the application program supports terminals in terminal class 4, this field must be written before any QTPUT call. Programs supporting application-to-application connections can also use this field but it will have no effect. This field can contain the following values:

0 The last or only block of the message. The application program will not call QTPUT again for the current connection until a QTGET call has returned an input block.

1 An intermediate block in a multiple block message. The application program will call QTPUT again for the current connection before a call to QTGET has returned an input block from that connection.

The connection involved in the current QTPUT call is identified in the connection-number field. QTRM uses the int-msg field to change the abt field of the application block header involved in the QTPUT call. If int-msg = 0, abt = 2; if int-msg = 1, abt = 1.

Figure E-1. Network Information Table Format (Sheet 6 of 9)

next-application-name	This 42-bit character data field contains the network application program name identifying the program to which a terminal should be switched during processing of a QTENDT call. This field can contain the following: 0 — The network software uses prompting dialog to determine the next application program the terminal communicates with, or disconnects the terminal if that is an appropriate action. NVF command — The Network Validation Facility reinitiates the login sequence for the terminal or causes terminal disconnection. valid program name — The terminal is switched to the indicated program without prompting dialog, when the switch is possible. If either the NVF command or valid program name option is used, the name placed in the field must be one to seven display code letters or digits, left-justified with blank fill within the field, and the first character must be alphabetic. If the NVF command option is used, the following commands are valid: BYE, LOGOUT — Cause the terminal to be disconnected from the host. HELLO, LOGIN — Reinitiate login for the terminal; if dialog is possible and required, the login prompting sequence begins. If the valid program name option is used, the name placed in the field must be the element name used to define the referenced application program in the system common deck COMTNAP. The QTOPEN call sets this field to zero. The application program must set or change this field as appropriate before each QTENDT call. Guidelines for the use of this field can be found in the discussion of voluntary connection termination under Terminating Connections in section 3. This field does not apply to QTENDT calls for application-to-application connections.
requested-application-name	This 42-bit character data field contains the network application program name identifying the program to which the current application program is requesting a connection with a QTLINK call.
terminal-name-i/ application-name-i	This 42-bit character data field contains the display code characters of the name used to identify the terminal or application program on connection i within the network. The name is one to seven letters or digits long and is left-justified with blank fill within this field. A terminal name used is obtained from the network configuration file entry for the terminal.
tclass-i	This 6-bit field contains the integer terminal class associated by the network with the terminal on connection i. The integer used in the field is one of those described for the tc field of the connection-request supervisory message presented in section 3. The integer is changed during a QTGET call whenever the terminal user has entered a TIP command to change the terminal class of the terminal on connection i. This field does not apply to application-to-application connections.
page-width-i	This 12-bit field contains the integer page width value associated by the network with the terminal on connection i. The integer used in the field has the significance explained in sections 2 and 3. The integer is changed during a QTGET call whenever the terminal user has entered a TIP command to change the page width or terminal class of the terminal on connection i. This field does not apply to application-to-application connections.
family-name-i	This 42-bit character data field contains the display code characters of the permanent file family name associated by the network with the terminal on connection i. The family name is one to seven letters or digits long and is left-justified with blank fill within this field. The name used is obtained from the login information for the terminal.

Figure E-1. Network Information Table Format (Sheet 7 of 9)

	This field does not apply to application-to-application connections.
dev-type-i	This 6-bit field contains one of the following integer values to identify the type of connection on connection i: 0 This connection is a terminal-to-application connection. 5 This connection is an application-to-application connection.
page-length-i	This 12-bit field contains the integer page length value associated by the network with the terminal on connection i. The integer used in the field has the significance explained in sections 2 and 3. The integer is changed during a QTGET call after the terminal user enters a TIP command to change the page length or terminal class of the terminal on connection i. This field does not apply to application-to-application connections.
user-name-i	This 42-bit character data field contains the display code characters of the NOS user name associated by the network with the terminal on connection i. The user name is one to seven letters, digits, or asterisks long and is left-justified with blank fill within the field. The name used is obtained from the login information for the terminal. This field does not apply to application-to-application connections.
res	Reserved by CDC. Currently unused.
block-size-i	This 12-bit field contains the integer block size in character units for the terminal on connection i. This block size is based on the network configuration file information for the terminal. The block size is a suggested value for adjusting the current-trans-size field based on efficiency considerations for the site. This field does not apply to application-to-application connections.
abl-i	This 6-bit integer field contains the number of transmission blocks permitted by the network to be in transit to connection i at a given moment. This block limit is based on the network configuration file information for the connection. The value used in this field determines the number of QTPUT calls that can be made on connection i before a QTGET call returns an indication that a block was delivered to the connection. a typical value is 2 for a terminal-to-application connection and 7 for an application-to-application connection.
current-abn-i	This 18-bit integer field contains the binary block number assigned by QTRM to the block sent to connection i by the last QTPUT call involving that connection. Every block sent by QTRM is assigned a number; the number assigned is sequential within the blocks sent to a given connection, and the sequence is restarted each time a new connection is assigned to the connection number.
acknowledged-abn-i	This 18-bit integer field contains the binary block number assigned by QTRM to the block last acknowledged on connection i. QTRM updates this field during a QTGET call, when QTRM determines that a block-delivered message has been received.
state-i	This 6-bit field contains the integer flag identifying the current processing state of connection i. This field has the values: 0 This connection number is currently not in use. 1 This connection is currently in a transition state while a new connection is being established. No other information in the associated 10-word entry for this connection should be considered accurate. 2 This connection is in use and in a normal state for input or output processing by the application program. 4 This connection is currently in a transition state while a new connection is being established. No other information in the associated 10-word entry for this connection should be considered accurate. This value is used for application-to-application connections only.

Figure E-1. Network Information Table Format (Sheet 8 of 9)

current-abl-i	This 6-bit integer field contains the number of sequential QTPUT calls that currently can be made for connection i without waiting for acknowledgment of delivery to the terminal or application. QTRM updates this field during QTGET and QTPUT calls, and the application program should examine the field before making a QTPUT call involving the connection. The values used in this field range from 0 to the value contained in the abl-i field; a value of 0 indicates that no blocks currently can be sent (the maximum number of blocks are in transit to the terminal).
upline-abh-i	This 60-bit field contains the binary application block header received by QTRM with the last input data block delivered by a QTGET call for connection i. This field has the format and contains the information described in section 2.
downline-abh-i	This 60-bit field contains the binary application block header created by QTRM to send with the last output data block involved in a QTPUT call for connection i. This field has the format and contains the information described in section 2.

Figure E-1. Network Information Table Format (Sheet 9 of 9)

For the convenience of programmers using COBOL 5.2 or subsequent versions that permit manipulation of information in 6-bit bytes, the fields within the network information table are defined in 6-bit byte multiples. The first occurrence of each field within figure E-1 indicates the type and size of the COBOL data item needed to define the field properly. These indications have the form I(x) or C(y), where I indicates binary integer data, C indicates character data, x indicates the number of bits comprising the integer, and y indicates the number of 6-bit display-code characters comprising the character string.

SUBROUTINES

Calls to the subroutines comprising QTRM do not contain many parameters because most communication between an application program and QTRM occurs through the fields in the network information table. The format of the subroutine calls conforms to the general guidelines given for the compiler-language form of the AIP routines, as described in sections 4 and 5. The QTRM routines reside in the libraries NETIO and NETIOD. These libraries are accessed as described in sections 4 and 6.

The format of the subroutine calls is given in the following subsections. Because QTRM is designed to be COBOL-oriented, the subroutine descriptions are COBOL-oriented. As described in section 4, QTRM can be used by programs written in languages other than COBOL.

INITIATING NETWORK ACCESS (QTOPEN)

The application program begins communication with the network by calling QTOPEN. This call has the format shown in figure E-2.

ENTER FORTRAN-X QTOPEN USING net-info-table

net-info-table	An input parameter, specifying the symbolic address for word 1 in the global portion of the network information table that should be used by QTRM during access to the network. In a COBOL call, this parameter is the Data Division descriptor for a level 01 data item containing level 02 or lower level data items in the form described in figure E-1. The fields in the network information table must be initialized before the call to QTOPEN is issued.

Figure E-2. QTOPEN Statement COBOL Call Format

QTOPEN is normally called only once per network communication access but can be called again after a QTCLOSE call. No QTRM call other than QTCLOSE can be made before QTOPEN is called. The call to QTOPEN performs the following functions:

Identifies to QTRM the address of the network information table defined by the application program

Allows QTRM to use the information already placed in the network information table by the application program

Allows QTRM to initialize the connection entry portions of the network information table and to store its own information in the global portion of the table

Causes QTRM to identify the application program to the network

Before QTOPEN is called, the application program must place information in the following fields of the table:

Application-name

Char-set

Num-conns

A-to-A

During processing of the call, QTRM uses this information to make appropriate AIP calls. For example, suppose the application program makes the following call:

```
ENTER FORTRAN-X QTOPEN USING NIT
```

where NIT is the network information table symbolic address and contains the application-name RMV2, the num-conns value of 5, and the char-set value of 4. In the Data Division of the program code, NIT appears as:

```
WORKING-STORAGE SECTION.
01  NIT.
02  GLOBAL.
03  APPLICATION-NAME PIC X(7) VALUE IS
        "RMV2".
03  CHAR-SET PIC 9 COMP-4 VALUE IS 4.
03  NUM-TERMS PIC 99 COMP-4 VALUE IS 5.
03  FILLER X(30).
    .
    .
    .
```

QTRM then connects the program to the network. QTRM identifies the program as the network application program called RMV2, supporting five terminals simultaneously on connections numbered 1 through 5 and using 6-bit display code for all input and output transmissions, and capable of processing transmissions.

When the QTOPEN call is completed, the application program either performs processing not related to network communication or uses the QTGET call and the sleep field of the network information table to suspend its processing until a terminal or application requests access to it. As soon as a terminal connection is completed (as soon as the state field in a connection entry of the network information table changes to 2), the program must identify itself to the terminal by sending a message to it using a call to QTPUT.

SENDING DATA (QTPUT)

The application program sends data through the network by calling QTPUT. This call has the format shown in figure E-3.

ENTER FORTRAN-X QTPUT USING ta-out-acn$_i$

ta-out-acn$_i$	An input parameter, specifying the symbolic address of the output text area for the terminal or application using connection acn$_i$. In a COBOL call, this parameter is the Data Division descriptor for a level 01 data item with a length defined by the max-trans-size value in the network information table. Data contained in ta-out-acn$_i$ is subject to the same constraints as character mode data in the text area used by any NETPUT call to AIP. These constraints are described in section 2.

Figure E-3. QTPUT Statement COBOL Call Format

Before making a call to QTPUT, the application program must perform the following operations:

Check the connection entry in the network information table to which the QTPUT call applies. The current-abl and/or state field must contain values that permit the call to be made.

Ensure that the connection number identifying the connection to which the call applies is in the connection-number field of the network information table.

Place a 1 in the int-msg field of the network information table if that action is necessary. This field must be used to service a terminal in terminal class 4 correctly when output queuing is performed. Terminals in that class, such as the 2741, have lockable keyboards. When output begins, the network software locks the terminal keyboard. The keyboard remains locked until a block is delivered that has an int-msg value of 0 associated with it. Then the keyboard is unlocked and no more output to the terminal is permitted until input is completed. If a message comprising nine blocks is being sent to the terminal, the first eight must have the int-msg field set to 1 to prevent the network software from interpreting an intermediate portion of a message (a single block) as the entire message and prohibiting output of the remainder of the blocks. The last block of a message must always have the int-msg field set to 0 before it is sent.

Place the data to be transmitted by the call into the text area identified by the parameter to be used in the call.

For terminal-to-application connections, place a unit separator code as a line terminator at the end of the data in the text area, if char-set is not 6-bit display code. QTRM will supply the final zero-byte terminator for 6-bit display code data for terminal-to-application connections (this QTRM function is described in more detail under the heading QTRM Formatting of Display-Coded Output).

Place the size of the current transmission in the global portion of the network information table. All embedded line terminators of either type must be included in the character count comprising the current transmission size. If a char-set field value other than 4 is used, any final unit separator must also be included in the character count; if a char-set field value of 4 is used, the character count should not include the zero-byte line terminator that QTRM supplies automatically for terminal-to-application connections.

Place the correct value in the max-trans-size field of the network information table, if that information was not stored there before a previous QTRM call. The max-trans-size value can be changed before any QTPUT call, because the output text area used for the call can be changed. QTRM uses the value in this field to determine the starting point of any backward scanning it is required to perform.

When the QTPUT call is completed, the data block involved in the call usually is in transit through the network, but is not necessarily already delivered to the connection. Delivery of the block, and the possibility of additional QTPUT calls for the same terminal, can be tracked through QTGET calls and the fields of the connection entry in the network information table.

QTRM sometimes cannot transmit a block through the network when a QTPUT call is made. After return from the QTPUT call, the application program should check the return-code field of the network information table to determine whether the block was actually transmitted.

As an example of QTPUT use, suppose an application program wants to send the message WELCOME ABOARD to the terminal on connection 1. The program sends the prompting message with a call such as that shown in the following statement set:

```
MOVE " WELCOME ABOARD " TO OUT-TEXT.
MOVE 1 TO CONNECTION-NUMBER.
MOVE 15 TO CURRENT-TRANS-SIZE.
ENTER FORTRAN-X QTPUT USING OUT-TEXT.
IF RETURN-CODE NOT = 0 GO TO PROBLEM.
```

Elsewhere in the program, the Data Division contains:

```
01 OUT-TEXT PIC X(100).
```

The Procedure Division also contains statements to test the entry for connection 1 to see whether the call can be made. These tests are necessary even for the first transmission from the program because QTRM might indicate that the connection is in a state temporarily preventing any transmission.

Use of the current-trans-size field and the first character of the text area during display-coded transmissions is described later in this section. The tests of the table needed before the QTPUT call are associated with the QTGET call and are described in the subsection on Output Queuing Using QTRM.

OBTAINING DATA OR CONNECTION STATUS (QTGET)

The application program obtains input from a connection or status information about a connection by calling QTGET. This call has the format shown in figure E-4.

ENTER FORTRAN-X QTGET USING ta-in-acn$_i$

ta-in-acn$_i$	An input parameter, specifying the symbolic address of the input text area for the connection acn$_i$. In a COBOL call, this parameter is the Data Division descriptor for a level 01 data item with a length defined by the max-trans-size value in the network information table. Data contained in ta-in-acn$_i$ is subject to the same constraints as character mode data in the text area used by any NETGET, NETGETF, NETGETL, or NETGTFL call to AIP. These constraints are described in section 2.

Figure E-4. QTGET Statement COBOL Call Format

Before making a call to QTGET, the application program must perform the following operations:

Place the correct value in the sleep field (in word 5) of the network information table, if that information was not stored there before a previous QTRM call. The sleep field value used can be changed before any QTGET call, if necessary.

Place the correct value in the max-trans-size field of the network information table, if that information was not stored there before a previous QTRM call. The max-trans-size value can be changed before any QTGET call, because the input text area used for the call can be changed.

During the QTGET call, QTRM updates all connection entry fields in the network information table for which information is available. This updating is performed for all connections, even though the call returns information concerning a single connection.

After updating the connection entry fields, the QTGET call causes the network to select a specific terminal or application for the program and QTRM to service. If no current requirement for servicing exists, QTRM either returns control to the program and places a value of 1 in the return-code field of the network information table, or suspends program execution until a servicing requirement arises. Whether return from the QTGET call is immediate or delayed depends on the sign and value of the sleep field when the call occurs.

If a servicing requirement exists, QTRM returns control to the program with information in one of two forms. If QTRM detects a network or connection status condition corresponding to a nonzero return-code field value, the return-code and connection-

number fields are appropriately set, and control returns to the program. QTRM does not deliver input data on return from such a call. Only status information is returned; any data queued for delivery to the program must be obtained through subsequent QTGET calls.

If QTRM does not detect a network or connection status condition corresponding to a nonzero return-code field value, the return-code field is set to zero. Control returns to the program after the connection-number field has been set to identify the connection affected by the call, and a single input block from that connection is delivered.

After return from a QTGET call, the application program should perform the following operations:

Check the return-code field of the network information table to determine if information or status was returned by the call

Check the connection-number field of the network information table to determine which connection is involved with the information or status returned in the call

Take an action appropriate for the return-code field value resulting from the call

Depending on the sleep value used when making the call and on events in the network, the response from any QTGET call can be any of the following:

Nothing (no data block, no status, and no connection entry changes).

No input data block, but one or more connection entries are updated to reflect delivery of previously sent blocks to the corresponding connections; the current-abl and acknowledged-abn fields are updated to reflect such deliveries.

An input data block and connection entry fields are updated.

An input data block but no connection entry fields are changed.

Status information (indication of a new connection, a user-break, or other status change on an existing connection) and connection entry fields are updated.

Status information (shutdown in progress, block discarded, and so forth) but no connection entry fields are changed.

The action taken by the application program after a QTGET call must not assume that only one of these conditions is possible, and should not exclude any of them. Use of the sleep field value and the updating of information in the connection entries is described further under Output Queuing Using QTRM later in this appendix.

The following example of QTGET use permits an application program to suspend its operation when there is no input for it to process, to process any input that does exist, and to perform any processing related to changes in the status of the network or of a specific terminal:

```
MOVE -1 TO SLEEP.
ENTER FORTRAN-X QTGET USING IN-DATA.
IF RETURN-CODE NOT = 0, GO TO STATUS-
    CHECK.
PERFORM  PROCESS-INPUT.
```

On return from the QTGET call, the application program either has data to process because the return-code field is 0 or else has status changes to process because the return-code field is not 0. If the return-code field contains a 0, the data block returned as a result of the QTGET call is found in the text area called IN-DATA.

The actions required by an application program for a specific nonzero return-code field value are described in the field definition information given previously in this appendix. The interaction of the QTGET calls and the fields in the network information table are the primary processing control mechanism of any application program using QTRM. If the QTGET call returns data and the character set is display code, QTRM blank fills the last line of the message if necessary to make the message end on a word boundary.

LINKING AN APPLICATION TO ANOTHER APPLICATION (QTLINK)

The application program requests a connection to another application program for the purpose of performing message transfers between them. This call has the format shown in figure E-5.

```
ENTER FORTRAN-X QTLINK
```

Figure E-5. QTLINK Statement COBOL Call Format

Before making a call to QTLINK, the application program must perform the following operations:

Set the A-to-A field in the network information table to 1. This field must be set before the program issues the call to QTOPEN.

Place the name of the application program to which connection is requested in the requested-application-name field in the network information table.

On return from the QTLINK request, the application program should check the value in the return-code field of the network information table. The return-code is interpreted from figure E-1 as follows:

A return-code value of 0 indicates that the request is being forwarded to NAM. The connection has not yet been completed.

A return-code value of 12 indicates that the request is ignored because another request for an application-to-application connection is outstanding and not yet complete (only one connection request can be outstanding at a time). The request should be retried at a later time.

After a call to QTLINK, a call to QTGET returns a value of 13 or 14 in the return-code field of the network information table. This completes the outstanding request for an application-to-application connection. The return-code field is interpreted from figure E-1 as follows:

A return code value of 14 indicates that the connection to the requested application has been made. The connection-number field of the network information table contains the connection number for the application-to-application connection.

A return code value of 13 indicates that the request for connection has been rejected. The sec-return-code field of the network information table contains the reason code returned from the request-for-connection supervisory message issued for the application-to-application connection.

ENDING A SINGLE TERMINAL OR APPLICATION CONNECTION (QTENDT)

The application program ends communication with a single terminal or application by calling QTENDT. This call has the format shown in figure E-6.

```
ENTER FORTRAN-X QTENDT
```

Figure E-6. QTENDT Statement COBOL Call Format

Before making a call to QTENDT, the application program should perform the following operations:

Place the connection number for the connection to be disconnected in the connection-number field of the global portion of the network information table.

Send a disconnection indicator message to the terminal or application so that the operator or application program does not attempt input.

Set the next-application-name field to zero or place an appropriate name or NVF command in it if the connection is to a terminal.

Check the connection entry in the network information table to determine if the current-abl field contains the same value as the abl field. Unless the values in these two fields are the same, at least one block of data remains undelivered to the terminal and QTENDT should not be called to end communication with the connection.

After a call to QTENDT is made, no additional information can be sent to the connection involved. Except for the state field, information contained in the connection entry portion of the network information table remains unchanged until the connection number associated with that entry is reassigned by the network software to another connection.

A call to QTENDT is not necessary to end a connection that has already been broken by events in the network. A call to QTENDT for a broken connection performs no action. A forced shutdown condition (a return-code field value of 10) is equivalent to a QTCLOSE call because QTRM automatically ends all connections without action by the application program.

As an example of QTENDT use, consider the following situation. The application program receives a command on connection number 4 that indicates the terminal user wants to end communication with the program. The program checks the fields in the connection entry of the network information table and determines that no blocks remain undelivered from previous QTPUT calls. Because the terminal user has requested that communication be ended, the program does not send a block to indicate that action. Instead, the following code is executed:

```
MOVE 4 TO CONNECTION-NUMBER.
ENTER FORTRAN-X QTENDT.
```

Upon return from the QTENDT call, connection number 4 becomes available for assignment by the network software to a connection with a new connection serviced by the program. The program therefore executes code that cleans up any remaining information in other tables or buffers concerning the old terminal on connection 4, so that no confusion exists if another terminal is assigned to the same number.

ENDING COMMUNICATION WITH THE NETWORK (QTCLOSE)

The application program can end communication with all connected terminals or applications and with the network software simultaneously by calling QTCLOSE. This call has the format shown in figure E-7.

```
ENTER FORTRAN-X QTCLOSE
```

Figure E-7. QTCLOSE Statement COBOL Call Format

The application program should call QTCLOSE only once after a QTOPEN call and cannot call any other QTRM routines except QTOPEN after calling QTCLOSE. Multiple calls to QTCLOSE have no effect. The program should always call QTCLOSE as part of its processing termination, unless a forced shutdown occurs. When a forced shutdown occurs (indicated by a return-code field value of 10), QTRM automatically ends all program access to the network.

A call to QTCLOSE performs the following operations:

Breaks all remaining connections (the terminal receives an APPLICATION FAILED message from the network software)

Ends program access to the network and makes new connections impossible

Closes the AIP debug log file and statistics file, if those files are being created

The QTCLOSE call is usually issued after one of the following situations arises:

- The program receives a shutdown or idledown indication from the network software (indicated by a return-code field value of 9).
- The program detects a specific clock time.
- The program receives a shutdown command from a terminal user or a connected application program.

Before making a QTCLOSE call, the application program should perform the following operations:

- Send a shutdown advisory message to all terminals and applications still connected to the program
- Determine that all transmitted blocks have been delivered to the connection
- Issue QTENDT calls for all remaining terminal connections so that APPLICATION FAILED messages do not appear at those connections

A QTCLOSE example complying with these recommendations would be too complex for the purposes of this appendix. Examples of QTCLOSE calls appear in several contexts within the same program at the end of this appendix.

SENDING A SYNCHRONOUS SUPERVISORY MESSAGE (QTTIP)

The application program can send a synchronous supervisory message through the network by calling QTTIP. The call format for QTTIP is identical to QTPUT. The message can be in character type 2 or 3 format.

If the application program sends a synchronous supervisory message which has a response (CTRL/CHAR/N or CTRL/RTC/R), the response is delivered when the application program calls QTGET. The application block type field of the upline-abh-i field identifies the data as a supervisory block. Supervisory message responses are always returned to the application program as character type 3.

OUTPUT FORMATTING AND EDITING

Output transmitted through QTRM to a terminal always uses the format effector feature of the AIP interactive virtual terminal interface. This format effector feature is completely described in section 2, and summarized in the following subsection.

Output transmitted through QTRM to another application within the same host need not be restricted to formatting conventions of the AIP Interactive Virtual Terminal interface. Both application programs must be prepared to handle data that passes between them. The length of the output block is based on the character set used, indicated in the char-set field, and is the value stored in the field named current-trans-size.

If display-coded output is transmitted to a terminal (a char-set field value of 4 is used), QTRM automatically performs editing functions on the contents of the text area used. These functions include placement of the final line terminator (zero-byte terminator) at the end of the output block, and determination of the number of characters in the block.

The current-trans-size field for blocks sent on application-to-application connections should be set to a value equal to the number of central memory words in the block using the character type specified in the char-set field. The contents of a block are not edited. If the data is in display-code (the char-set field is equal to 4) and the current-trans-size field is equal to zero, the effective current-trans-size is determined by scanning the output text area.

FORMAT EFFECTORS

The network software assumes that the first character of each line in a block sent to a terminal through QTRM begins with a format effector character. The format effector character controls placement of the line on the terminal output mechanism in a manner similar to the way a carriage control character functions in output sent to a batch line printer. Format effector characters are discarded by the network software, so an application program should always format its output to prevent the first character of data from being interpreted erroneously as a format effector character.

EDITING DISPLAY-CODE OUTPUT

Each block sent by a QTPUT call can contain one or many lines of data. Each line of data must end with a line terminator byte appropriate to the value in the char-set field of the network information table. The terminator must follow the last character position in the line.

When an application program uses a char-set field value of 4, it must allow 12 to 66 bits of text area buffer space for the final 12-bit zero-byte line terminator. For COBOL programs, this means the text area used for any QTPUT call must be at least 11 characters longer than the longest block of data to be sent.

Generating the zero-byte terminator at the appropriate location in the text area is difficult in a COBOL program. QTRM therefore always generates the last such byte required by the block during its processing of a QTPUT call (interim line terminators must still be generated by the application program before the call).

If an output block contains only one line, QTRM can be used as follows to perform all output formatting required:

- The program sets the current-trans-size field of the network information table to 0.
- The program blank-fills the entire output text area to be used and then places the block to be sent into the text area (the block must include the format effector character). The block must contain at least one character other than a blank.

The program calls QTPUT. QTRM then determines where the text area ends by examining the max-trans-size field of the network information table. QTRM scans backward through the output text area, skipping over blanks until it encounters a nonblank character. QTRM inserts the zero-byte terminator after this character, then calculates the number of characters in the block and transmits it through the network.

This option eliminates unnecessary trailing blanks from the last output line of any block and makes it unnecessary for the application program to calculate how many characters are being transmitted. An alternate method permits transmission of trailing blanks, as follows:

The program places the output block containing at least one character (the format effector character) in the output text area.

The program places the number of characters comprising the block in the current-trans-size field of the network information table.

The program calls QTPUT. QTRM scans forward the indicated number of character positions, writes the final zero-byte terminator, if necessary, after the last character counted, and transmits the block. QTRM adjusts the character count indicating the block length to compensate for the line terminator bytes it has added.

Both options require that the last character in the block not be a colon or consecutive colons, in character positions 9 and 10 of a central memory word. Two consecutive colons might be misinterpreted as a zero-byte terminator on a system using a 64-character set.

QTRM (QTPUT) always adds a terminator for 6-bit display code data. If the program provides its own final line terminator for display-coded output, QTRM does not function in the same manner as it does for output transmissions occurring with a char-set field value of 2 or 3. No automatic terminator placement occurs during a QTPUT call involving those char-set field values.

OUTPUT QUEUING USING QTRM

Application programs commonly need to transmit more than one block in a message. If all of the terminals serviced by the program have large values assigned for the abl parameter, no special programming is required. Most networks, however, use small values for the abl parameter. When a program using QTRM executes in such a network, it must use an output queue to store blocks ready for output whenever the network does not permit immediate output of them.

An output queue processor using QTRM can be coded according to the algorithm shown in figure E-8. This algorithm uses the sleep field parameter in the global portion of the network information table and depends on use of the current-abl parameter in the terminal entry portion of the table. The following paragraphs explain the logic used to design the algorithm.

When an application program services only one terminal, the network can be made to cope with situations where the program produces output faster than a terminal can reproduce the output. The program sets the sleep parameter to a positive integer, and the network simply rolls the program out of central memory until the terminal catches up with the program.

You cannot use the sleep parameter as a solution when the application program services more than one terminal because the program is always rolled back in when input is available from any terminal. Thus, input from terminal B brings the program back into central memory even though the output backlog for terminal A has not disappeared. A program servicing several terminals always requires an output queuing algorithm that applies to each, when each terminal potentially can be sent more than one block in a single message.

Programs can also be coded for the opposite (type-ahead) environment, when the terminal user wants to enter many input messages and receive only one output transmission. Input queuing and support of typeahead are not discussed in this manual. Type-ahead can be supported without any interaction of an application program with the network protocol.

The primary control variable of the output queuing algorithm is the terminal connection number. Both the accompanying flow chart and the sample progam code depend on the use of the connection number field in conjunction with the terminal entry fields of the network information table during the output queue scanning process.

An application program can control the flow of its output to a specific terminal by checking the current-abl field of the terminal entry in the network information table before each QTPUT call involving that terminal. If the field contains a zero, the call cannot be made without violating network protocol; if the field does not contain a zero, the QTPUT call can be made.

The current-abl, acknowledged-abn, and other fields in the network information table are only updated by QTRM during processing of a QTGET call. Tests of these fields are not meaningful unless a QTGET call is made before the tests. To properly control output flow, the application program must make periodic calls to QTGET with a positive value in the sleep field of the network information table, regardless of whether the program expects input from a terminal. The size of the positive value is a tuning consideration determined by such things as the average length of output blocks and the speed of the terminal being serviced.

These QTGET calls return control to the program after the sleep period. The program can then test the current-abl field and make any QTPUT calls that have become feasible. A QTPUT call is feasible whenever the current-abl value is nonzero. If the value is zero, another QTGET call must be made.

An application program can use two forms of output flow control queuing. The program can actually generate all output required as a response to one input, then queue the output in large internal buffers or disk files. This queued output is then spooled to the terminal in QTPUT calls involving one or more lines in blocks up to the max-block-

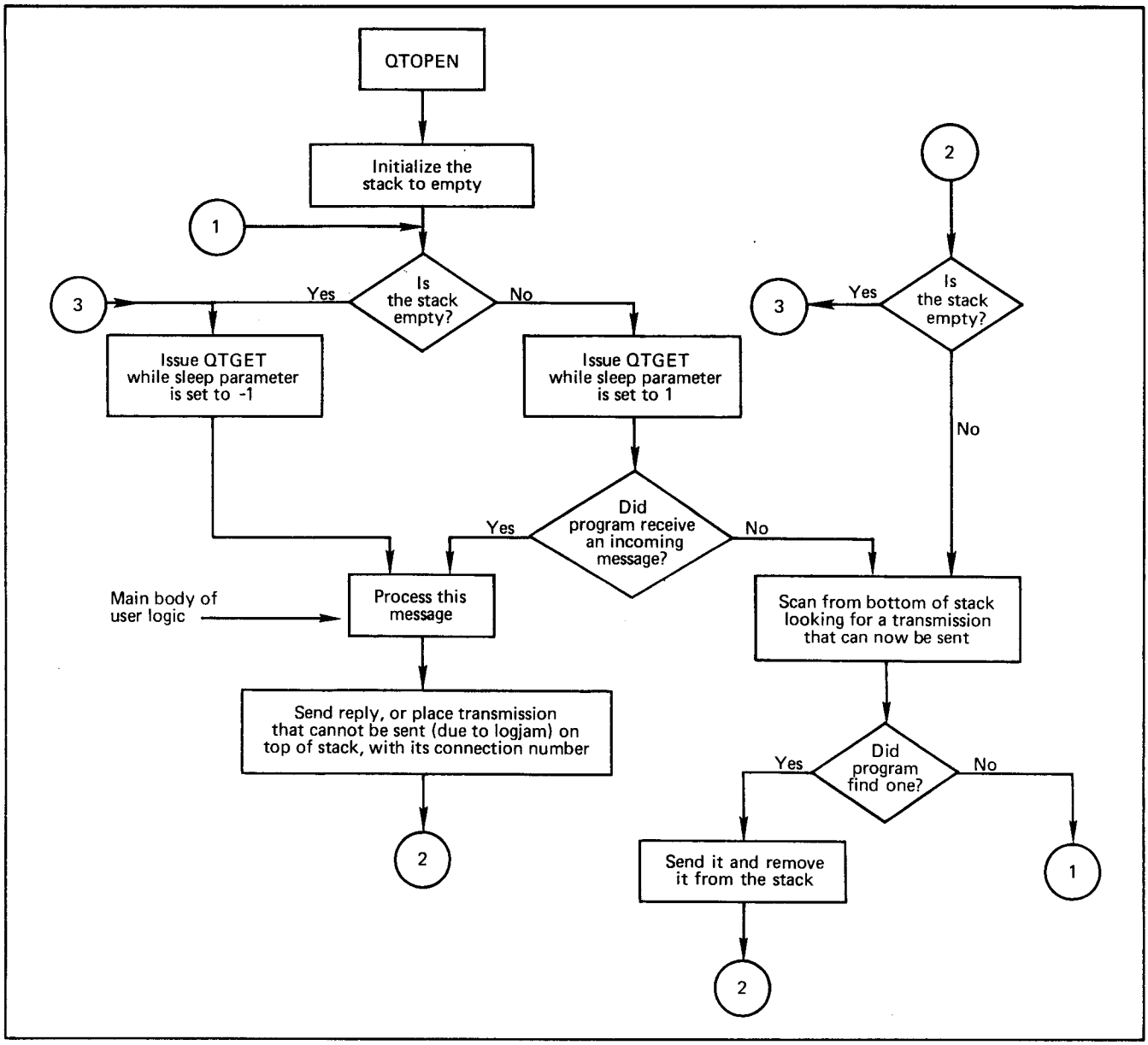


Figure E-8. Algorithm for Output Buffering Using QTRM

size value for the terminal entry in the network information table. The algorithm already described is used to control the occurrence of the QTPUT calls.

Alternatively, the application program can queue its input requests. When the flow control algorithm described previously shows that a QTPUT call can be made, the program can generate only enough output for one QTPUT call. After making the call, an uncompleted input request is returned to the queue to await additional processing the next time the flow control algorithm permits another QTPUT call for the terminal. This approach requires a small input queue for each terminal, but does not require large internal buffers for output storage.

The second approach minimizes field length requirements and mass storage access requirements for the program. Also, the program can avoid wasted output processing when the terminal user issues a user-break to terminate output after only one or two blocks of the output have been delivered. With the first approach, processing for the remainder of the output has already occurred and is wasted. With the second approach, no processing for the additional output occurred and therefore the additional processing can be bypassed.

SAMPLE PROGRAM

Figure E-9 contains the source code listing for a COBOL program that demonstrates use of QTRM in the simplest form possible. Program ECHO-RMV2 is similar to the FORTRAN program RMV3 shown in appendix I. Both programs return to the terminal user each line entered from the terminal. Both programs queue output blocks and permit a separate prompting message to be output after each returned message line. Both programs acknowledge entry of a user-break character with dialog. Both programs shut down operation after receiving a terminal operator command.

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM     AOPT= 66/CDC/CDCS2

 1               IDENTIFICATION DIVISION.
 2               PROGRAM-ID. ECHO-RMV2.
 3               ENVIRONMENT DIVISION.
 4               CONFIGURATION SECTION.
 5               INPUT-OUTPUT SECTION.
 6               FILE-CONTROL.
 7               DATA DIVISION.
 8               FILE SECTION.
 9               WORKING-STORAGE SECTION.
10               01 NETWORK-INFORMATION-TABLE.
11                   02 GLOBAL-PORTION.
12                     03 APPLICATION-NAME PIC X(7).
13                     03 CHARACTER-SET PIC 9 COMP-4.
14                     03 NUMBER-TERMINALS PIC 999 COMP-4.
15                     03 NAM-SUPERVISOR-WORD PIC X(10).
16                     03 FILLER PIC X(20).
17              *
18              *THE PICTURE SIZE USED FOR COMPUTATIONAL ITEMS SUCH AS
19              *MAX-TRANS-SIZE AND SLEEP IS CHOSEN TO PERMIT STORAGE OF
20              *THE LARGEST POSSIBLE FIELD VALUE WITHOUT TRUNCATION OF
21              *THE VALUE DIGITS.
22              *
23                     03 MAX-TRANS-SIZE PIC 999 COMP-4.
24                     03 MESSAGE-LENGTH PIC 999 COMP-4.
25                     03 SLEEP PIC S9 COMP-4.
26                     03 CONNECTION-NUMBER PIC 999 COMP-4.
27                     03 RETURN-CODE PIC 9 COMP-4.
28                     03 SECONDARY-RETURN-CODE PIC 9 COMP-4.
29                     03 INTERMEDIATE-MESSAGE PIC 9 COMP-4.
30                     03 NEXT-APPLICATION-NAME PIC X(7).
31                     03 FILLER PIC X(43).
32                   02 TERMINAL-ENTRY OCCURS 5 TIMES.
33                     03 TERMINAL-NAME PIC X(7).
34                     03 TERMINAL-CLASS PIC 9 COMP-4.
35                     03 PAGE-WIDTH PIC 999 COMP-4.
36                     03 FAMILY-NAME PIC X(7).
37                     03 FILLER PIC X.
38                     03 PAGE-LENGTH PIC 999 COMP-4.
39                     03 USER-NAME PIC X(7).
40                     03 FILLER PIC X.
41                     03 MAXIMUM-BLOCK PIC 999 COMP-4.
42                     03 ABL PIC 9 COMP-4.
43                     03 CURRENT-ABN PIC 9(4) COMP-4.
44                     03 ACKNOWLEDGED-ABN PIC 9(4) COMP-4.
45                     03 STATE PIC 9 COMP-4.
46                     03 FILLER PIC X.
47                     03 CURRENT-ABL PIC 9 COMP-4.
48                     03 FILLER PIC X(10).
49                     03 UPLINE-ABH PIC X(10).
50                     03 DOWNLINE-ABH PIC X(10).
51                     03 FILLER PIC X(30).
52               01 INCOMING.
53                   02 COMMAND PIC X(20).
54                   02 REST-OF-DATA PIC X(80).
55               01 OUTGOING.
56                   02 PRINT-CONTROL PIC X.
57                   02 OUT-MESSAGE PIC X(120).
58               01 FOUND-FLAG PIC 9.
59               01 QUEUE-SIZE PIC 99.
60               01 HOLDING-QUEUE.

          COLUMN   1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 1 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM     AOPT= 66/CDC/CDCS2


61              *
62              *THIS IS A PUSHDOWN QUEUE USED FOR  STORAGE OF THOSE
63              *OUTPUT BLOCKS THE PROGRAM IS TEMPORARILY PREVENTED FROM SENDING
64              *TO THE TERMINAL BECAUSE OF BLOCK LIMIT OR OTHER EVENTS IN THE
65              *NETWORK.
66              *
67                   02 QUEUE-ENTRY OCCURS 1 TO 60 TIMES DEPENDING ON QUEUE-SIZE
68                       INDEXED BY INX-1 INX-2.
69                       03 S-CONNECTION-NUMBER PIC 999 COMP-4.
70                       03 S-MESSAGE PIC X(120).
71                       03 S-INTERMEDIATE-MESSAGE PIC 9 COMP-4.
72
73
74
75               PROCEDURE DIVISION.
76
77
78               INITIALIZATION.
79              *
80              *HERE, THE NETWORK INFORMATION TABLE IS PRESET.
81              *
82                   MOVE #RMV2# TO APPLICATION-NAME.
83                   MOVE 4 TO CHARACTER-SET.
84                   MOVE 100 TO MAX-TRANS-SIZE.
85              *
86              *THE FORMAT EFFECTOR CHARACTER #.# CAUSES THE CURSOR TO
87              *RETURN TO THE LEFT EDGE OF THE SCREEN OR PAGE
88              *FOLLOWING THE CONTENTS OF OUT-MESSAGE. THIS ACTION
89              *LEAVES THE CURSOR POSITIONED SO THAT THE USER CAN ENTER
90              *A LINE EQUAL TO THE FULL PAGE WIDTH OF THE TERMINAL.
91              *
92
93                   MOVE #.# TO PRINT-CONTROL.
94                   MOVE SPACES TO OUT-MESSAGE.
95                   MOVE SPACES TO INCOMING.
96                   MOVE 5 TO NUMBER-TERMINALS.
97                   MOVE -1 TO SLEEP.
98                   MOVE 1 TO INTERMEDIATE-MESSAGE.
99                   MOVE 0 TO QUEUE-SIZE.
100                  MOVE 0 TO FOUND-FLAG.
101                  ENTER FORTRAN-X QTOPEN USING NETWORK-INFORMATION-TABLE.
102
103             *
104             *ALL TERMINALS WILL BE SWITCHED AUTOMATICALLY TO IAF
105             *WHEN THEY ARE DISCONNECTED FROM THIS PROGRAM.
106             *
107                  MOVE #IAF# TO NEXT-APPLICATION-NAME.
108
109              MAIN-LOOP.
110                  PERFORM RECEIVER THRU RECEIVE-EXIT.
111
112                  IF STATE (CONNECTION-NUMBER) = 1
113                     GO TO MAIN-LOOP.
114                  IF RETURN-CODE = 2
115                     MOVE 0 TO INTERMEDIATE-MESSAGE
116                     PERFORM DISPLAY-BANNER THRU BANNER-EXIT
117                     GO TO MAIN-LOOP.
118                  IF RETURN-CODE = 4
119                     PERFORM PUSH-DOWN-QUEUE
120                     GO TO MAIN-LOOP.
121                  IF RETURN-CODE = 6
122                     PERFORM CONNECTION-BROKEN-ROUTINE THRU CB-EXIT
123                     GO TO MAIN-LOOP.


          COLUMN   1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 2 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM      AOPT= 66/CDC/CDCS2

124                 IF RETURN-CODE = 7 OR = 8
125                    PERFORM FLUSH-QUEUE
126                    MOVE 0 TO INTERMEDIATE-MESSAGE
127                    MOVE ≠.≠ TO PRINT-CONTROL
128                    MOVE ≠NO ACTION TAKEN. NEXT ENTRY↓≠ TO OUT-MESSAGE
129                    PERFORM SENDER THRU SEND-EXIT
130                    GO TO MAIN-LOOP.
131                 IF RETURN-CODE = 9
132                    GO TO WRAP-UP.
133              *
134              *TO SIMPLIFY THE PROGRAM, ONLY EXPECTED CONDITIONS ARE PROCESSED
135              *BY THE PRECEDING CODE. ALL OTHER CONDITIONS CAUSE THE PROGRAM
136              *TO PLACE A DIAGNOSTIC MESSAGE IN THE FILE CALLED OUTPUT (WITH
137              *THE DISPLAY STATEMENT) AND SHUT DOWN. NO DIAGNOSTIC APPEARS AT
138              *THE TERMINAL.
139              *
140                   IF RETURN-CODE NOT = 0
141                      DISPLAY ≠PROGRAM BUG OR OTHER ERROR≠ RETURN-CODE ≠ ≠
142                      SECONDARY-RETURN-CODE STOP RUN.
143
144
145
146
147                      MOVE ≠.≠ TO PRINT-CONTROL.
148
149
150              *
151              *IF A TERMINAL USER ENTERS THE WORD END, THE USER IS
152              *DISCONNECTED BUT THE PROGRAM CONTINUES TO SERVICE OTHER
153              *TERMINALS.
154              *
155                   IF COMMAND = ≠END≠
156                      PERFORM END-CONNECTION THRU EC-EXIT
157                      GO TO MAIN-LOOP.
158              *
159              *IF A TERMINAL USER ENTERS THE WORD SHUTDOWN, THE USER IS
160              *DISCONNECTED AND THE PROGRAM SHUTS DOWN.
161              *
162                   IF COMMAND = ≠SHUTDOWN≠
163                     MOVE 0 TO INTERMEDIATE-MESSAGE
164                     MOVE ≠.≠ TO PRINT-CONTROL
165                     MOVE ≠BYE FOREVER↵≠ TO OUT-MESSAGE
166                     PERFORM SENDER THRU SEND-EXIT
167
168                     GO TO WRAP-UP.
169
170              *
171              *THE FOLLOWING CODE BEGINS THE INPUT-ECHOING PORTION
172              *OF THIS PROGRAM.
173              *
174                   MOVE INCOMING TO OUT-MESSAGE
175                   MOVE 1 TO INTERMEDIATE-MESSAGE
176                   MOVE ≠.≠ TO PRINT-CONTROL
177                   PERFORM SENDER THRU SEND-EXIT
178              *
179              *SEND PROMPT FOR NEXT LINE, WHICH ALSO ENDS PRESENT OUTPUT
180              *MESSAGE TO THIS TERMINAL.
181              *
182                   MOVE 0 TO INTERMEDIATE-MESSAGE
183                   MOVE ≠.≠ TO PRINT-CONTROL
184                   MOVE ≠NEXT ENTRY↓≠ TO OUT-MESSAGE
185                   PERFORM SENDER THRU SEND-EXIT
186                   GO TO MAIN-LOOP.

        COLUMN   1         2         3         4         5         6         7         8
        12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 3 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM     OPT= 66/CDC/CDCS2

187             *
188             *THIS ENDS THE MAIN PROGRAM PORTION OF ECHO-RMV2. THE FOLLOWING
189             *PARAGRAPHS COMPRISE THE SUBROUTINES USED BY THE MAIN PROGRAM.
190             *
191
192
193
194              RECEIVER.
195                   IF QUEUE-SIZE = 0
196                      MOVE -1 TO SLEEP
197             *
198             *THE NEXT LINE PREVENTS LEFTOVER CHARACTERS FROM THE END OF THE
199             *LAST INPUT LINE FROM BEING INCLUDED IN THE TRANSFER OF THE
200             *CURRENT (AND PRESUMABLY SHORTER) LINE.
201             *
202                      MOVE SPACES TO INCOMING
203                      ENTER FORTRAN-X QTGET USING INCOMING
204                      GO TO RECEIVE-EXIT.
205                   MOVE 1 TO SLEEP
206                   MOVE SPACES TO INCOMING
207                   ENTER FORTRAN-X QTGET USING INCOMING.
208                   IF RETURN-CODE NOT = 1
209                      GO TO RECEIVE-EXIT
210                      ELSE NEXT SENTENCE.
211              QUEUE-OUTPUT-CODE.
212                   MOVE 0 TO FOUND-FLAG.
213                   PERFORM QUEUE-SCAN VARYING INX-1 FROM 1 BY 1
214                      UNTIL FOUND-FLAG = 1 OR INX-1 EXCEEDS QUEUE-SIZE.
215                   IF FOUND-FLAG = 0
216                      GO TO RECEIVER
217                      ELSE NEXT SENTENCE.
218                   SET INX-1 DOWN BY 1.
219             *
220             *THE REMAINING CODE ATTEMPTS TO REMOVE ALL
221             *QUEUED OUTPUT FROM THE OUTPUT QUEUE, ONE ENTRY AT A
222             *TIME, REGARDLESS OF CONNECTION NUMBER. EACH SEND
223             *OPERATION IS FOLLOWED BY A RETURN TO THE POINT IN
224             *THE PROGRAM WHERE STATUS UPDATES ARE OBTAINED.
225             *
226                   MOVE S-INTERMEDIATE-MESSAGE (INX-1) TO INTERMEDIATE-MESSAGE.
227                   MOVE S-CONNECTION-NUMBER (INX-1) TO CONNECTION-NUMBER.
228                   IF STATE (CONNECTION-NUMBER) = 3 GO TO RECEIVE-EXIT.
229                   MOVE *.* TO PRINT-CONTROL.
230                   MOVE S-MESSAGE (INX-1) TO OUT-MESSAGE.
231                   PERFORM QUEUE-COMPRESSION VARYING INX-2 FROM INX-1 BY 1
232                      UNTIL INX-2 = QUEUE-SIZE.
233                   SUBTRACT 1 FROM QUEUE-SIZE.
234                   PERFORM SENDER THRU SEND-EXIT.
235                   IF QUEUE-SIZE = 0
236                      GO TO RECEIVER
237                      ELSE GO TO QUEUE-OUTPUT-CODE.
238              RECEIVE-EXIT.
239                   EXIT.
240
241
242
243
244
245              QUEUE-SCAN.
246                   MOVE S-CONNECTION-NUMBER (INX-1) TO CONNECTION-NUMBER.
247                   IF CURRENT-ABL (CONNECTION-NUMBER) EXCEEDS 0
248                      MOVE 1 TO FOUND-FLAG.
249
250

         COLUMN   1         2         3         4         5         6         7         8
         12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 4 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM     ADPT= 66/CDC/CDCS2


251              QUEUE-COMPRESSION.
252                  MOVE QUEUE-ENTRY (INX-2 + 1) TO QUEUE-ENTRY (INX-2).
253
254
255              FLUSH-QUEUE.
256                  SET INX-1 INX-2 TO 1.
257                  PERFORM FLUSH-LOOP UNTIL INX-2 EXCEEDS QUEUE-SIZE.
258                  SET INX-1 DOWN BY 1.
259                  SET QUEUE-SIZE TO INX-1.
260              FLUSH-LOOP.
261                  IF S-CONNECTION-NUMBER (INX-1) = CONNECTION-NUMBER
262                     SET INX-2 UP BY 1
263                  ELSE
264                     PERFORM CONDITIONAL-QUEUE-MOVE
265                     SET INX-1 INX-2 UP BY 1.
266              CONDITIONAL-QUEUE-MOVE.
267                  IF INX-1 NOT = INX-2
268                     MOVE QUEUE-ENTRY (INX-2) TO QUEUE-ENTRY (INX-1).
269
270
271
272
273              SENDER.
274                  IF CURRENT-ABL (CONNECTION-NUMBER) = 0
275                     PERFORM PUSH-DOWN-QUEUE GO TO SEND-EXIT.
276
277
278             *
279             *THE PROGRAM HAS QTRM SCAN BACKWARDS THROUGH THE MESSAGE
280             *AREA AND TRUNCATE THE MESSAGE AUTOMATICALLY. THIS PROCEDURE
281             *IS COMPARABLE TO THE ONE USED BY CYBER RECORD MANAGER FOR
282             *Z-TYPE RECORDS.
283             *
284                  MOVE 0 TO MESSAGE-LENGTH.
285                  ENTER FORTRAN-X QTPUT USING OUTGOING.
286             *
287             *IF NAM HAS STOPPED OUTPUT ON THE CONNECTION TEMPORARILY, OR IF
288             *THE BLOCK LIMIT HAS BEEN EXCEEDED (AN EVENT THAT SHOULD NOT
289             *HAPPEN) FOR THE CONNECTION, THE MESSAGE IS RETURNED TO THE
290             *QUEUE FOR A LATER TRY.
291             *
292                  IF RETURN-CODE  = 5 PERFORM PUSH-DOWN-QUEUE.
293              SEND-EXIT.
294                  EXIT.
295
296
297
298
299
300              PUSH-DOWN-QUEUE.
301                  ADD 1 TO QUEUE-SIZE.
302                  IF QUEUE-SIZE EXCEEDS 60 DISPLAY *QUEUE OVERFLOW ABORT*
303                     PERFORM DUMPER VARYING INX-1 FROM 1 BY 1
304                     UNTIL INX-1 EXCEEDS 60
305                     STOP RUN.
306
307                  MOVE INTERMEDIATE-MESSAGE TO S-INTERMEDIATE-MESSAGE
308                     (QUEUE-SIZE).
309                  MOVE CONNECTION-NUMBER TO S-CONNECTION-NUMBER (QUEUE-SIZE).
310                  MOVE OUT-MESSAGE TO S-MESSAGE (QUEUE-SIZE).
311
312
313

         COLUMN   1         2         3         4         5         6         7         8
         12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 5 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM     AOPT= 66/CDC/CDCS2


314             *
315             *THE FOLLOWING PROMPT IS  MANDATORY, BECAUSE QTRM DOES NOT
316             *AUTOMATICALLY ISSUE A PROMPT TO A NEW
317             *CONNECTION TO INITIALIZE THAT CONNECTION. THE FOLLOWING
318             *PROMPT IS SENT BECAUSE GOOD PROGRAMMING PRACTICE
319             *REQUIRES A NETWORK APPLICATION PROGRAM TO IDENTIFY ITSELF
320             *TO A TERMINAL USER.
321             *
322              DISPLAY-BANNER.
323                  MOVE ≠.≠ TO PRINT-CONTROL.
324                  MOVE ≠THIS IS RMV2 USING QTRM. ENTER SOMETHING.≠ TO
325                     OUT-MESSAGE.
326                  PERFORM SENDER THRU SEND-EXIT.
327              BANNER-EXIT.
328                  EXIT.
329
330
331
332             *
333             *NO CALL TO QTENDT IS NECESSARY DURING THIS PROCESSING BRANCH,
334             *BECAUSE QTRM AUTOMATICALLY CLEANS UP THE CONNECTION WHEN IT
335             *RETURNS THE CONNECTION-BROKEN STATUS.
336             *
337              CONNECTION-BROKEN-ROUTINE.
338                  DISPLAY ≠CONNECTION BROKEN - TERMINAL USER HUNG UP ≠
339                     CONNECTION-NUMBER
340                  DISPLAY ≠ FAMILY ≠ FAMILY-NAME (CONNECTION-NUMBER)
341
342                  DISPLAY ≠ USER ≠ USER-NAME (CONNECTION-NUMBER).
343              CB-EXIT.
344                  EXIT.
345
346
347
348
349
350
351
352
353
354
355             *
356             *THE WAIT-FOR-QUIET CALLS PROVIDE A DELAY LOOP FOR THE
357             *NETWORK TO CLEAN UP ALL OUTSTANDING SUPERVISORY MESSAGE
358             *TRAFFIC RELATED TO THE SHUTDOWN. WITHOUT THIS LOOP,
359             *SOME TERMINAL CONNECTIONS WOULD RECEIVE AN
360             *≠APPLICATION FAILED≠ MESSAGE.
361             *
362              WRAP-UP.
363                  PERFORM GRACEFUL-DISCONNECTS THRU GD-EXIT VARYING
364                     CONNECTION-NUMBER
365                     FROM 1 BY 1 UNTIL CONNECTION-NUMBER = 6.
366                  ENTER FORTRAN-X QTCLOSE.
367                  STOP RUN.
368
369
370




COLUMN   1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 6 of 7)

```
CDC COBOL 5.3 - LEVEL 577       SOURCE LISTING OF ECHO-RM      AOPT= 66/CDC/CDCS2

371             GRACEFUL-DISCONNECTS.
372                 IF STATE (CONNECTION-NUMBER) = 2 PERFORM FLUSH-QUEUE
373                    MOVE 0 TO INTERMEDIATE-MESSAGE
374                    MOVE #.# TO PRINT-CONTROL
375                    MOVE #SHUTDOWN COMING# TO OUT-MESSAGE
376                    PERFORM SENDER THRU SEND-EXIT
377                    ENTER FORTRAN-X OTENDT.
378             GD-EXIT.
379                 EXIT.
380
381
382
383             END-CONNECTION.
384                 PERFORM FLUSH-QUEUE
385                 MOVE 0 TO INTERMEDIATE-MESSAGE
386                 MOVE #.# TO PRINT-CONTROL
387                 MOVE #GOODBYE FOR NOW..# TO OUT-MESSAGE.
388                 PERFORM SENDER THRU SEND-EXIT.
389                 ENTER FORTRAN-X OTENDT.
390             EC-EXIT.
391                 EXIT.
392
393
394
395
396
397
398             DUMPER.
399
400                 DISPLAY S-CONNECTION-NUMBER (INX-1)
401                    S-MESSAGE (INX-1).



         COLUMN   1         2         3         4         5         6         7         8
         12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure E-9. Sample Program ECHO-RMV2 Source Listing (Sheet 7 of 7)

Figure E-10 contains a dayfile listing showing the control statements used to execute ECHO-RMV2. ECHO-RMV2 exists as an indirect access source code file named RMV2.

Figure E-11 contains a complete debug log file listing for a single execution of ECHO-RMV2. This log file is very similar to the one shown in appendix I for program RMV3 because both programs use essentially the same AIP routines for the same functions and support the same kind of dialog. Figure E-12 contains a statistics file listing for ECHO-RMV2.

Figure E-13 is a terminal printer listing for two sequential terminal sessions using ECHO-RMV2 during a single execution of that program. The listing includes program-generated messages and a terminal input message that is echoed back.

```
09.22.27.QTRMEXP.
09.22.27.UCCR, 76, 041,        0.414KCDS.
09.22.27.USER([redacted])
09.22.27.CHARGE([redacted])
09.22.27.SETPR(70)
09.22.28.COBOL5.
09.22.40.0677008 CM,     1.688 CPS, 0000008 ECS
09.22.40.LDSET(LIB=NETIOD)
09.22.40.LGO.
09.22.45.NAM  VER 1.2-  4E
09.36.07.REWIND,ZZZZZSN.
09.36.07.COPY,ZZZZZSN.
09.36.07. EOI ENCOUNTERED.
09.36.08.DLFP(I=0)
09.36.08. DLFP COMPLETE
09.36.08.COPY,INPUT,QTRMEXP.
09.36.09. EOI ENCOUNTERED.
09.36.09.REWIND,QTRMEXP.
09.36.09.SAVE,QTRMEXP.
09.36.09.UEAD,        0.002KUNS.
09.36.09.UEPF,        0.009KUNS.
09.36.09.UEMS,        6.429KUNS.
09.36.09.UECP,        2.070SECS.
09.36.09.AESR,        7.329UNTS.
09.36.09.$OUT(*/OP=E)
09.36.09. NO FILES PROCESSED.
09.36.09.$DAYFILE(OUTPUT,JT=D)
09.36.41.UCLP, 76, 046,       1.024KLNS.
```

Figure E-10. Sample Program ECHO-RMV2 Job Dayfile Listing

```
12.26.41.000        NETON   (004272)     ANAME = RMV2          DATE = 79/05/07          MSG NO. 000001
 NSUP ADDR = 001442  MINACN =00001  MAXACN =00005



12.26.44.458        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000002
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0010

     001 630000001400200 30600000000120001000  X#   AP H   C
     002 50D71B16DB44800 24153433055555044000  TM10E  D5     Q M H
     003 000000000000104 00000000000000000404          DD
     004 000000000000000 00000000000000000000
     005 [redacted]                                    M
     006 [redacted]                                    3 Q    4
     007 [redacted]
     008 [redacted]
     009 [redacted]
     010 [redacted]                                        X R


12.26.44.458        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000003
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 634000001400101 30640000000120000401  X"   AP DA  C


12.26.47.074        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000004
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830700001000000 40603400000100000000  5#1  A
```

Figure E-11. Debug Log File Listing for ECHO-RMV2 (Sheet 1 of 5)

```
12.26.47.075        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000005
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 834700001000000 40643400000100000000  5"1  A       G


12.26.47.076        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000006
 ABT =02  ADR =0001  ABN =000001  ACT =04  STATUS = 00000000   TLC = 0050

     001 BD42094ED253B52 57241011235511235522  .THIS IS R   B N S
     002 35676D55324E1ED 15263555252311160755  MV2 USING   #VV S$
     003 45448DBED14E505 21242215575505162405  QTRM. ENTE  ED   NP
     004 4AD4CF34550824E 22552317150524101116  R SOMETHIN  T L EP N
     005 1EF000000000000 07570000000000000000  G.


12.26.49.624        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000007
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000040 40601000000100000100  5#H  A  A


12.27.04.867        NETGETL (005464)  ALN =0001  HA =003242  TA =001535  TLMAX =0010      MSG NO. 000008
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0043

     001 50816D385614B45 24100555160530245505  THE NEXT E  P M8V
     002 394499B494ED06D 16242231551123550155  NTRY IS A   S I IN M
     003 55315298248504B 25230522460222050113  USER-BREAK  U1R $
     004 99CB43201481OD4 46345503100122010324  -1 CHARACT     2 H
     005 152BC0000000000 05225700000000000000  ER.          +


12.27.04.893        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000009
 ABT =01  ADR =0001  ABN =000002  ACT =04  STATUS = 00000000   TLC = 0050

     001 BD4205B4E15852D 57241005551605302455  .THE NEXT    B   XR
     002 14E51266D253B41 05162422315511235501  ENTRY IS A   NQ&M%;A
     003 B554C54A6092141 55252305224602220501   USER-BREA   T J
     004 2E672D0C8052043 13463455031001220103  K-1 CHARAC    R    C
     005 5054AF000000000 24052257000000000000  TER.        PT


12.27.04.895        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000010
 ABT =02  ADR =0001  ABN =000003  ACT =04  STATUS = 00000000   TLC = 0020

     001 BCE15852D14E512 57160530245505162422  .NEXT ENTR    XR NQ
     002 679000000000000 31710000000000000000  Y?          &Y


12.27.06.980        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000011
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000080 40601000000100000200  5#H  A  B


12.27.06.981        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000012
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 83020000100000C0 40601000000100000300  5#H  A  C


12.27.27.110        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000013
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 83000100100000C0 40600001000100000300  5# A A  C
```

Figure E-11. Debug Log File Listing for ECHO-RMV2 (Sheet 2 of 5)

```
12.27.27.110        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000014
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830100001000000 40600400000100000000  5#D  A


12.27.27.112        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000015
 ABT =02  ADR =0001  ABN =000004  ACT =04  STATUS = 00000000   TLC = 0040

     001 BCE3ED0435093CE 57161755010324111716  .NO ACTION      5 ,
     002 B5404B14EBED385 55240113051657551605   TAKEN. NE   T  N
     003 614B45394499E40 30245505162422317100 . XT ENTRY?   AKE9D
     004 000000000000000 00000000000000000000


12.27.28.694        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063     MSG NO. 000016
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000100 40601000000100000400  5#H  A  D


12.27.44.305        NETGETL (005464)  ALN =0001  HA =003242  TA =001535  TLMAX =0010     MSG NO. 000017
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0042

     001 50816D385614B45 24100555160530245505  THE NEXT E  P M8V
     002 394499B494ED06D 16242231551123550155  NTRY IS A   S I IN M
     003 553152982485O4B 25230522460222050113  USER-BREAK  U1R $
     004 99DB432014810D4 46355503100122010324  -2 CHARACT     2 H
     005 152000000000000 05220000000000000000  ER


12.27.44.310        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000018
 ABT =01  ADR =0001  ABN =000005  ACT =04  STATUS = 00000000   TLC = 0050

     001 BD4205B4E15852D 57241005551605302455  .THE NEXT    B   XR
     002 14E51266D253B41 05162422315511235501  ENTRY IS A   NQ&M%;A
     003 B554C54A6092141 55252305224602220501   USER-BREA   T J
     004 2E676D0C8052043 13463555031001220103  K-2 CHARAC    V    C
     005 505480000000000 24052200000000000000  TER         PT


12.27.44.312        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000019
 ABT =02  ADR =0001  ABN =000006  ACT =04  STATUS = 00000000   TLC = 0020

     001 BCE15852D14E512 57160530245505162422  .NEXT ENTR    XR NQ
     002 679000000000000 31710000000000000000  Y?          &Y


12.27.46.413        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063     MSG NO. 000020
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000140 40601000000100000500  5#H  A  E


12.27.46.414        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063     MSG NO. 000021
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000180 40601000000100000600  5#H  A  F


12.27.49.581        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063     MSG NO. 000022
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830002001000180 40600002000100000600  5# B A  F


12.27.49.582        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000023
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830100001000000 40600400000100000000  5#D  A
```

Figure E-11. Debug Log File Listing for ECHO-RMV2 (Sheet 3 of 5)

```
12.27.49.584        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000024
 ABT =02  ADR =0001  ABN =000007  ACT =04  STATUS = 00000000   TLC = 0040

     001 BCE3ED0435093CE 57161755010324111716  .NO ACTION      5 ,
     002 B5404B14EBED385 55240113051657551605   TAKEN. NE   T  N
     003 614B45394499E40 30245505162422317100  XT ENTRY?   AKE9D
     004 000000000000000 00000000000000000000


12.27.52.221        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000025
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 8302000010001C0 40601000000100000700  5#H  A  G


12.27.54.961        NETGETL (005464)  ALN =0001  HA =003242  TA =001535  TLMAX =0010      MSG NO. 000026
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0003

     001 14E100000000000 05160400000000000000  END


12.27.54.963        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000027
 ABT =02  ADR =0001  ABN =000008  ACT =04  STATUS = 00000000   TLC = 0020

     001 BC73CF102645B46 57071717040231055506  .GOODBYE F   S  &E
     002 3D2B4E3D7BEF000 17225516172757570000  OR NOW..    C


12.27.54.964        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000028
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 C00000001000000 60000000000100000000  #    A


12.27.54.964        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000029
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002

     001 630600001000000 30603000000100000000  X#X  A     C
     002 2411ADB6DB40000 11010655555555000000  IAF         A [M4


12.27.55.893        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000030
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000200 40601000000100001000  5#H  A  H


12.27.57.454        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000031
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 634600001000000 30643000000100000000  X"X  A      CF


12.28.05.278        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000032
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0010

     001 630000001400200 30600000000120001000  X#   AP H   C
     002 50D71B16DB44800 24153433055555044000  TM10E  D5     Q M H
     003 000000000000104 00000000000000000404          DD
     004 000000000000000 00000000000000000000
     005 [redacted]                                    M
     006                                               3 Q    4
     007
     008
     009
     010                                                  X R
```

Figure E-11. Debug Log File Listing for ECHO-RMV2 (Sheet 4 of 5)

```
12.28.05.278        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000033
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 634000001400101 30640000000120000401  X"   AP DA  C


12.28.07.967        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000034
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830700001000000 40603400000100000000  5#1  A


12.28.07.967        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000035
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 834700001000000 40643400000100000000  5"1  A       G


12.28.07.971        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000036
 ABT =02  ADR =0001  ABN =000001  ACT =04  STATUS = 00000000   TLC = 0050

     001 BD42094ED253B52 57241011235511235522  .THIS IS R   B N S
     002 35676D55324E1ED 15263555252311160755  MV2 USING   #VV S$
     003 45448DBED14E505 21242215575505162405  QTRM. ENTE  ED   NP
     004 4AD4CF34550824E 22552317150524101116  R SOMETHIN  † L EP N
     005 1EF000000000000 07570000000000000000  G.


12.28.10.055        NETGET  (005451)  ACN =0000  HA =003242  TA =003270  TLMAX =0063      MSG NO. 000037
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000040 40601000000100000100  5#H  A  A


12.28.14.731        NETGETL (005464)  ALN =0001  HA =003242  TA =001535  TLMAX =0010      MSG NO. 000038
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0008

     001 4C855410F5CE000 23102524041727160000  SHUTDOWN    L T


12.28.14.741        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000039
 ABT =02  ADR =0001  ABN =000002  ACT =04  STATUS = 00000000   TLC = 0020

     001 BC2645B463D2156 57023105550617220526  .BYE FOREV   &E C
     002 152D80000000000 05226600000000000000  ER!         AR


12.28.14.744        NETPUT  (005767)   HA =003242   TA =001547                          MSG NO. 000040
 ABT =02  ADR =0001  ABN =000003  ACT =04  STATUS = 00000000   TLC = 0020

     001 BD32155043D73AD 57231025240417271655  .SHUTDOWN    2 PC
     002 0CF349387000000 03171511160700000000  COMING        4


12.28.14.745        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000041
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 C00000001000000 60000000000100000000  #    A


12.28.14.746        NETPUT  (005767)   HA =003242   TA =003270                          MSG NO. 000042
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002

     001 630600001000000 30603000000100000000  X#X  A      C
     002 2411ADB6DB40000 11010655555555000000  IAF         A [M4


12.28.15.723        NETOFF  (003267)       DATE =79/05/07                          MSG NO. 000043
```

Figure E-11. Debug Log File Listing for ECHO-RMV2 (Sheet 5 of 5)

```
NAM STATISTICS GATHERING STARTED
NETON   DATE  79/05/07.   TIME 12.26.41.
NAM STATISTICS GATHERING TERMINATED
NETOFF  DATE  79/05/07.   TIME 12.28.16.
CPU TIME USED:          0.098 SEC

NUMBER OF PROCEDURE CALLS
    NETGET                  38
    NETGETL                 22
    NETPUT                  21
    NETWAIT                 19
NUMBER OF WORKLIST TRANSFER ATTEMPTS
    SUCCESSFUL              63
NUMBER OF INPUT/OUTPUT BLOCKS TRANSFERRED
    INPUT   ABT=0           40
    INPUT   ABT=2            4
    INPUT   ABT=3           16
    OUTPUT  ABT=1            2
    OUTPUT  ABT=2            9
    OUTPUT  ABT=3           10
NUMBER OF ERRORS
```

Figure E-12. Statistics File Listing for ECHO-RMV2

```
THIS IS RMV2 USING QTRM. ENTER SOMETHING.
The next character is a user-break-1 character.
THE NEXT CHARACTER IS A USER-BREAK-1 CHARACTER.
NEXT ENTRY?
:
NO ACTION TAKEN. NEXT ENTRY?
The next character is a user-break-2 character.
THE NEXT CHARACTER IS A USER-BREAK-2 CHARACTER.
NEXT ENTRY?
)
NO ACTION TAKEN. NEXT ENTRY?
end
GOODBYE FOR NOW..
RMV2     CONNECT TIME 00.01.14.
TERMINAL:      40, NAMIAF
RECOVER/ CHARGE: charge,[illegible]
$CHARGE,[illegible]
/bye, rmv2

[illegible]  LOG OFF     15.38.17.
[illegible]  SRU       1.001 UNTS.


IAF      CONNECT TIME 00.00.24.
THIS IS RMV2 USING QTRM. ENTER SOMETHING.
shutdown
BYE FOREVER!
SHUTDOWN COMING
RMV2     CONNECT TIME 00.00.21.
TERMINAL:      35, NAMIAF
```

Figure E-13. ECHO-RMV2 Sample Dialog

INTRODUCTION

The network, through which terminals access a host and its applications (like TAF), can support many types of terminals. All supported terminals are grouped by the network into 17 standard terminal classes. Each terminal class has specific operating characteristics, also referred to as terminal characteristics. These terminal characteristics, taken together, make up a terminal definition and are predefined by the network to closely match the operating characteristics of actual terminals. The network accepts changes to any of these terminal definitions through terminal definition commands. These commands specify one or more parameters, each of which may give a new value to a terminal characteristic. Tables F-1 and F-2 list these terminal definitions, the default values, and possible values for each of the 17 classes. To effectively use a terminal class, choose a terminal class whose default terminal definition corresponds to the characteristics of your terminal. If you do not know the terminal class in use, you can enter the CH command (described in this appendix) to ascertain the terminal class and some of the characteristics of your terminal.

When a terminal logs in to the network, either the network assumes it to be of a certain terminal class or assigns it a terminal class that resembles the terminal's actual characteristics. In either case, if the characteristics of your terminal do not match those of its assigned terminal class, you can change the values of the terminal characteristics or even change the terminal class by using the terminal definition commands described in this appendix. You can enter terminal definition commands any time the terminal is connected to the network (refer to appendix H).

When you use a terminal definition command to change a value, that change remains in effect until the terminal is disconnected from the network or another terminal definition command is used to change the value. Even application switching and logout do not change the values of the terminal characteristics if the terminal has not been disconnected from the network. Only by disconnecting the terminal from the network (disconnecting the phone and redialing on dial-up terminals or turning a dedicated terminal off and then on again) or by entering the TC terminal definition command (described in this appendix) can you reset the values to their default values without redefining each characteristic individually.

It is also possible to change terminal definitions under the interactive access facility (IAF) using the TRMDEF command or by using control byte 16 from an interactive job. The NOS 2 Reference Set, Volume 3, describes the TRMDEF command under Terminal Definition Commands. Refer to the NOS 2 Reference Set, Volume 4, for information on control bytes.

It is also possible to change terminal definitions using the TERMDEF request in TAF. This command is discussed in appendix H. Refer to the Network Products Network Access Methods, Version 1 Network Definition Language Reference Manual for more information on terminal characteristics.

TERMINAL DEFINITION COMMAND FORMAT

This section describes the general format of a terminal definition command. The specific format of each command appears later in this appendix.

The general formats are:

ct keyword=value†

ct $keyword_1$=$value_1$...ct $keyword_n$=$value_n$†

ct

Represents the network control character for your terminal as defined by the CT terminal definition command. Table F-1 gives the default network control character for your terminal.††

keyword

Represents a mnemonic associated with a terminal characteristic.

value

Specifies a particular value associated with the terminal definition mnemonic. Table F-2 gives the range of values for each terminal definition keyword. On some commands the =value portion of the command is optional and defaults to =Y.

$keyword_i$

Same as keyword.

$value_i$

Same as value.

†The spaces shown are for clarity only. Do not include spaces when you enter a terminal definition command.

††You must press the ATTN key on a 2741-compatible terminal before entering this character.

TABLE F-1. DEFAULT TERMINAL DEFINITIONS

Mnemonic	Description	Terminal Types																
		TTY M33	CDC 713-10, 751-1, 756	IBM 2741	TTY M40-2	HAZ 2000	CDC 752	Tek 4000	X.25 any	HASP POST	200 UT	714-30	711-10	714-10, 20	HASP PRE	734	2780	3780
tc	Terminal Class	1	2	4	5	6	7	8	(1)	9	10	11	12	13	14	15	16	17
AB	Abort output block	CAN	CAN	(	CAN	CAN	CAN	CAN	na	na	na	na	na	na	na	na	na	na
BS	Backspace character	BS	BS	BS	none	BS	BS	BS	(1)	na	na	na	na	na	na	na	na	na
B1	User break 1	DLE	DLE	:	DLE	DLE	DLE	DLE	(1)	:	:	:	:	:	:	:	na	na
B2	User break 2	DC4	DC4	)	DC4	DC4	DC4	DC4	(1)	)	)	)	)	)	)	)	na	na
BR	Break as user break 1	N	N	na	N	N	N	N	(1)	na	na	na	na	na	na	na	na	na
CI	Carriage return idles	2	0	8	1	0	0	0	(1)	na	na	na	na	na	na	na	na	na
CN	Cancel character	CAN	CAN	(	CAN	CAN	CAN	CAN	(1)	(	(	(	(	(	(	(	na	na
CP	Cursor positioning	Y	Y	na	Y	Y	Y	Y	N	na	na	na	na	na	na	na	na	na
CT	Network control character	ESC	ESC	%	ESC	ESC	ESC	ESC	(1)	%	%	%	%	%	%	%	%	%
DL	Single-message transparent input mode																	
	Character delimiter	CR	CR	NL	CR	CR	CR	CR	(1)	na	(4)	(4)	(4)	(4)	na	(4)	na	na
	Character count delimiter	2043	2043	na	2043	2043	2043	2043	(1)	na	na	na	na	na	na	na	na	na
	Timeout delimiter	NO	NO	na	NO	NO	NO	NO	(1)	na	na	na	na	na	na	na	na	na
EB	End-of-block																	
	Character	EOT	EOT	na	EOT	EOT	EOT	EOT	(2)	na	(4)	(4)	(4)	(4)	na	(4)	na	na
	Cursor positioning	CL	CL	na	CL	CL	CL	CL	NO	na	NO	NO	NO	NO	na	NO	na	na
EL	End-of-logical line																	
	Character	CR	CR	NL	CR	CR	CR	CR	(1)	na	(5)	(5)	(5)	(5)	na	(5)	na	na
	Cursor positioning	LF	LF	na	LF	LF	LF	LF	NO	na	NO	NO	NO	NO	na	NO	na	na

TABLE F-1. DEFAULT TERMINAL DEFINITIONS (Contd)

Mnemonic	Description	Terminal Types																
		TTY M33	CDC 713-10, 751-1, 756	IBM 2741	TTY M40-2	HAZ 2000	CDC 752	Tek 4000	X.25 any	HASP POST	200 UT	714-30	711-10	714-10, 20	HASP PRE	734	2780	3780
EP	Echoplex mode	N	N	na	N	N	N	N	na	na	na	na	na	na	na	na	na	na
FA	Full-ASCII input mode	N	N	N	N	N	N	N	(1)	na	N	N	N	N	na	N	na	na
IC	Input device flow control	N	N	na	N	N	N	N	na	na	na	na	na	na	na	na	na	na
IN	Input device	KB	KB	KB	KB	KB	KB	KB	BK	na	BK	BK	BK	BK	na	BK	na	na
LI	Line feed idles	1	0	1	3	3	0	0	(1)	na	na	na	na	na	na	na	na	na
LK	Lockout unsolicited messages	N	N	N	N	N	N	N	(1)	NO	NO	NO	NO	NO	NO	NO	na	na
OC	Output device flow control	N	N	na	N	N	N	N	na	na	na	na	na	na	na	na	na	na
OP	Output device	PR	DI	PR	DI	DI	DI	DI	(1)	na	na	na	na	na	na	na	na	na
PA	Parity processing	E	E	na	E	E	E	E	(3)	na	na	na	na	na	na	na	na	na
PG	Page waiting	N	N	N	N	N	N	N	(1)	na	Y	Y	Y	Y	na	Y	na	na
PL	Page length	0	0	0	0	0	0	0	(1)	0	13	16	16	16	0	13	0	0
PW	Page width	72	80	132	74	74	80	74	(1)	80	80	80	80	80	80	80	80	120
SE	Special editing mode	N	N	N	N	N	N	N	na	na	na	na	na	na	na	na	na	na
XL	Multimessage transparent input mode	NO	NO	NO	NO	NO	NO	NO	NO	na	NO	NO	NO	NO	na	NO	na	na

Notes:

(1) Same as for the terminal if it were not connected via a packet-switching network.

(2) End-of-packet sequence (M bit is reset).

(3) The parity bit is ignored on input and is generated for output as for the basic terminal type.

(4) ETX (resulting from the SEND key).

(5) ESC (resulting from the CARRIAGE RETURN key).

TABLE F-2. PARAMETER RANGES FOR TERMINAL DEFINITION COMMANDS

Mnemonic	Description	Terminal Types																
		TTY M33	CDC 713-10, 751-1, 756	IBM 2741	TTY M40-2	HAZ 2000	CDC 752	Tek 4000	X.25 any	HASP POST	200 UT	714-30	711-10	714-10, 20	HASP PRE	734	2780	3780
tc	Terminal Class	1	2	4	5	6	7	8	(1)	9	10	11	12	13	14	15	16	17
AB	Abort output block	(1)	(1)	(1)	(1)	(1)	(1)	(1)	na	na	na	na	na	na	na	na	na	na
BS	Backspace character	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	na	na	na	na	na	na	na	na	na
B1	User break 1	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	na	na
B2	User break 2	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	na	na
BR	Break as user break 1	Y,N	Y,N	na	Y,N	Y,N	Y,N	Y,N	(2)	na	na	na	na	na	na	na	na	na
CI	Carriage return idles	0-99, CA	0-99, CA	0-99, CA	0-99, CA	0-99, CA	0-99, CA	0-99, CA	(2)	na	na	na	na	na	na	na	na	na
CN	Cancel character	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	na	na
CP	Cursor positioning	Y,N	Y,N	na	Y,N	Y,N	Y,N	Y,N	(2)	na	na	na	na	na	na	na	na	na
CT	Network control character	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(2)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	na	na
DL	Single-message transparent input mode																	
	Character delimiter	0-FF	0-FF	2D	0-FF	0-FF	0-FF	0-FF	(2)	na	03	03	03	03	na	03	na	na
	Character count delimiter	1-4095	1-4095	na	1-4095	1-4095	1-4095	1-4095	(2)	na	na	na	na	na	na	na	na	na
	Timeout delimiter	TO	TO	na	TO	TO	TO	TO	(2)	na	na	na	na	na	na	na	na	na
EB	End-of-block																	
	Character	(1), EL, EB	(1), EL, EB	na	(1), EL, EB	(1), EL, EB	(1), EL, EB	(1), EL, EB	(2)	na	EB	EB	EB	EB	na	EB	na	na
	Cursor positioning	(3)	(3)	na	(3)	(3)	(3)	(3)	(2)	na	NO	NO	NO	NO	na	NO	na	na

TABLE F-2. PARAMETER RANGES FOR TERMINAL DEFINITION COMMANDS (Contd)

Mnemonic	Description	Terminal Types															
		TTY M33	CDC 713-10, 751-1, 756	IBM 2741	TTY M40-2	HAZ 2000	CDC 752	Tek 4000	X.25 any	HASP POST	200 UT	714-30	711-10	714-10, 20	HASP PRE	734	2780	3780
EL	End-of-logical line																	
	Character	(1), EL, EB	(1), EL, EB	na	(1), EL, EB	(1), EL, EB	(1), EL, EB	(1), EL, EB	(2)	na	EB, EL	EB, EL	EB, EL	EB, EL	na	EB, EL	na	na
	Cursor positioning	(3)	(3)	na	(3)	(3)	(3)	(3)	NO	na	NO	NO	NO	NO	na	NO	na	na
EP	Echoplex mode	Y,N	Y,N	na	Y,N	Y,N	Y,N	Y,N	na	na	na	na	na	na	na	na	na	na
FA	Full-ASCII input mode	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	(2)	na	Y,N	Y,N	Y,N	Y,N	na	Y,N	na	na
IC	Input device flow control	Y,N	Y,N	na	Y,N	Y,N	Y,N	Y,N	na	na	na	na	na	na	na	na	na	na
IN	Input device	(4)	(4)	KB,XK, X	(4)	(4)	(4)	(4)	X	na	X	X	X	X	na	X	na	na
LI	Line feed idles	0-99, CA	0-99, CA	0-99, CA	0-99, CA	0-99, CA	0-99 CA	0-99, CA	(2)	na	na	na	na	na	na	na	na	na
LK	Lockout messages	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	(2)	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	na	na
OC	Output device flow control	Y,N	Y,N	na	Y,N	Y,N	Y,N	Y,N	na	na	na	na	na	na	na	na	na	na
OP	Output device	(5)	(5)	PR,DI	(5)	(5)	(5)	(5)	PR,DI	na	na	na	na	na	na	na	na	na
PA	Parity processing	Z,O, E,N	Z,O, E,N	na	Z,O, E,N	Z,O, E,N	Z,O, E,N	Z,O, E,N	(6)	na	Y,N	Y,N	Y,N	Y,N	na	Y,N	na	na
PG	Page waiting	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	(2)	0	0,8-255	0,8-255	0,8-255	0,8-255	0	0,8-255	0	0
PL	Page length	0,8-255	0,8-255	0,8-255	0,8-255	0,8-255	0,8-255	0,8-255	(2)	0	0,20	0,20	0,20	0,20	0,20	0,20	0,20	0,20
PW	Page width	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	(2)	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255	0,20-255

TABLE F-2. PARAMETER RANGES FOR TERMINAL DEFINITION COMMANDS (Contd)

Mnemonic	Description	Terminal Types																
		TTY M33	CDC 713-10, 751-1, 756	IBM 2741	TTY M40-2	HAZ 2000	CDC 752	Tek 4000	X.25 any	HASP POST	200 UT	714-30	711-10	714-10, 20	HASP PRE	734	2780	3780
SE	Special editing mode	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	Y,N	na	na	na	na	na	na	na	na	na	na
TC	Terminal class input	1,2, 5-8	1,2, 5-8	4	1,2, 5-8	1,2, 5-8	1,2, 5-8	1,2, 5-8	1,2, 5-8	9	10,15	11-13	11-13	11-13	14	10,15	16	17
XL	Multimessage transparent input mode																	
	Character delimiter	0-FF	0-FF	2D	0-FF	0-FF	0-FF	0-FF	(2)	na	03	03	03	03	na	03	na	na
	Character terminator	0-FF	0-FF	2D	0-FF	0-FF	0-FF	0-FF	(2)	na	na	na	na	na	na	na	na	na
	Timeout delimiter	TO	TO	na	TO	TO	TO	TO	na	na	na	na	na	na	na	na	na	na
	Character-count delimiter	1-4095	1-4095	na	1-4095	1-4095	1-4095	1-4095	(2)	na	na	na	na	na	na	na	na	na

Notes:

(1) Any character from the ASCII 128-character set except any lowercase or uppercase alphabetic character, any digit 0 through 9, NULL, SOH, STX, =, DEL, *or* space.

(2) Same as for the terminal if it were not connected via a packet-switching network.

(3) CR, LF, CL, NO.

(4) KB, PT, BK, X, XK, XP.

(5) PR, DI, PT.

(6) Same as for the terminal if it were not connected via a packet-switching network. Parity is ignored on input and is generated for output.

The second format allows you to make multiple terminal definitions in one line. If the line extends beyond character position 54, the results are unpredictable. Also, if you change the network control character ct in a concatenated group of commands, the CT command does not take effect until the remaining commands in the group have been processed.

As with all commands described in this manual, you must press the message transmission key in order for the system to process your command. Appendix G shows the default message transmission keys for the various terminal classes. This appendix describes how you can change the message transmission key (refer to the EB and EL commands).

Example:

To change the page width of a 200 User Terminal, enter these characters

```
%PW=50
```

and press the SEND key. The % character (the 200 User Terminal default) is the network control character, PW is the keyword associated with page width, 50 is the new page width for the terminal, and the SEND key is the message transmission key for a 200 User Terminal.

RESTRICTIONS

To enter a terminal definition command at asynchronous terminals from which you can enter more than one line before transmitting (IN=BK is specified), the terminal definition command must be the last input line in the transmission. If the input is from paper tape (IN=PT is specified), the tape reader must stop. Unpredictable results can occur if more input follows the terminal definition command in the transmission block.

You should not enter terminal definition commands while output is in progress. The network may not be able to perform such commands immediately.

The network presets all user-adjustable terminal characteristics any time you establish a connection to the network. The site administrators choose these preset values. Additionally, when a terminal is disconnected from the network and then reconnected, the network resets any terminal characteristics you have changed to their preset values.

Characters or character codes you specify for the value portion of terminal definition commands are subject to the following restrictions:

You cannot use the following ASCII characters (or their character codes) in the value portion of the AB, CN, BS, B1, B2, CT, EB, or EL commands:

```
Any lowercase or uppercase alphabetic
    character
Any digit 0 through 9
NUL
SOH
STX
=
DEL
Space
```

The ASCII characters or character codes you specify for the AB, BS, B1, B2, CN, CT, EB, and EL terminal definitions must all be different, except that AB and CN can have the same value, and EB and EL can have the same value.

TERMINAL DEFINITION COMMANDS

This subsection describes the function and format of each terminal definition command. The headings consist of the command name (a two-character mnemonic) followed by the terminal attribute it defines. Generally, the format descriptions do not show the default values or ranges of values since they vary with terminal class. Tables F-1 and F-2 show this information. If you are unable to find the key on your keyboard that corresponds to a particular character given in table F-1, consult the documentation for your terminal. This section refers to categories of terminal classes to which commands apply. The categories are as follows:

Asynchronous (TC = 1-8)

Mode 4 (TC = 10-13, 15)

HASP (TC = 9, 14)

Bisynchronous (TC = 16, 17)

PSN terminals, which are a subset of asynchronous terminals (TC = 1, 2, 5-8)

AB - ABORT OUTPUT BLOCK CHARACTER†

The AB command specifies the character used to abort an output block. When you enter this character from the terminal as the only character in a line, the current transmission block of output to the terminal is discarded.

The format is:

```
ct AB=ab
```

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

ab — Specifies the new abort output block character. This character must differ from the characters defined for BS, B1, B2, CT, EB, and EL.

This command applies only to asynchronous terminals.

†Not valid for terminals connected to packet-switching networks (PSNs).

AR - AUTOMATIC CHARACTER RECOGNITION†

The AR command allows you to change the terminal's character set or its line speed, in many cases, by means of a switch on the terminal or a removable type ball. You can enter this command any time after the system prompts you for input, even before you enter your user name during login.

The format is:

```
ct AR
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

After entering the AR command, you must physically select the terminal's character set (for example, change the type ball) and enter a carriage return. The system acknowledges the carriage return operation and indicates that the line speed recognition is complete by returning two line feeds. At this point, enter a right parenthesis so that the system can recognize the terminal's new character set. When the system recognizes the new character set, it issues a line feed and the message AR ACCEPTED.. to indicate you can continue. This is also the auto recognition procedure used to connect the terminal to the system (refer to appendix H).

This command applies to asynchronous terminals.

BR - BREAK KEY AS USER BREAK 1

The BR command determines if the BREAK key (interruption key for interactive output) also functions as the interruption sequence (user break 1 sequence).

The format is:

```
ct BR=option
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

option Determines if this feature is enabled or disabled. A value of Y enables and a value of N disables this feature.

This command applies only to asynchronous terminals.

BS - BACKSPACE CHARACTER

The BS command specifies the character used to delete the previous input character.

The format is:

```
ct BS=bs
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

bs Specifies the new backspace character. This character must differ from the characters currently defined for AB, B1, B2, CN, CT, EB, and EL.

It is possible to backspace only to the beginning of the current physical line; additional backspaces are disregarded. When a page width of 0 is selected, the network assumes a page width of 100 characters for backspacing.

This command applies only to asynchronous terminals.

B1 - INTERRUPTION CHARACTER

The B1 command specifies the character that, when entered as the only character in a logical line (interruption character followed by the message transmission key), causes program interruption.

The format is:

```
ct B1=ub1
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

ub1 Specifies the new user break 1 character. This character must differ from the characters currently defined for AB, BS, B2, CN, CT, EB, and EL.

The interruption sequence is also called the user break 1 sequence. This process is discussed in the NOS 2 Reference Set, Volume 3. The BR command determines whether the BREAK key can also function as the B1 character.

This command does not apply to bisynchronous terminals.

B2 - TERMINATION CHARACTER

The B2 command specifies the character that, when entered as the only character in a logical line (termination character followed by the message transmission key), causes program termination.

The format is:

```
ct B2=ub2
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

ub2 Specifies the new user break 2 character. This character must differ from the characters currently defined for AB, BS, B1, CN, CT, EB, and EL.

The termination sequence is also called the user break 2 sequence. This process is discussed in the NOS 2 Reference Set, Volume 3.

This command does not apply to bisynchronous terminals.

†Not valid for terminals connected to a PSN. Terminals connected to a PSN use the ASCII code set.

CH - DISPLAY TERMINAL CHARACTERISTICS

The CH command displays some of the terminal's current characteristics.

The format is:

```
ct CH
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

The CH command produces a display in the following format:

```
TC=tc,BS=bs,CN=cn,AB=ab,B1=ub1,B2=ub2,EL=el,EB=eb
```

where the variable portions appear as ASCII characters if printable, mnemonics (for example, STX) if nonprintable, or N/A if the attribute is not applicable.

CI - CARRIAGE RETURN IDLE COUNT

The CI command specifies the number of idle characters to be inserted into the output stream after a carriage return.

The format is:

```
ct CI=ci
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

ci Specifies the new carriage return idle count. This value can be from 0 to 99 or the string CA. A value of CA restores the carriage return idle count to the default value.

When the network produces a carriage return, the network outputs the specified number of idle characters before outputting the next line. This allows time for the carriage return function and ensures that characters are not lost because of printing attempts during the carriage return operation.

This command applies only to asynchronous terminals.

CN - CANCEL CHARACTER

The CN command specifies the character used to cancel the logical line currently being input.

The format is:

```
ct CN=cn
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

cn Specifies the new cancel character. This character must differ from the characters currently defined for BS, B1, B2, CT, EB, and EL.

When you enter the cancel character as the last character in a line, the entire logical line in progress is cancelled. The system responds to a cancel line character by printing *DEL* on the next line and positioning the carriage to the beginning of a new line.

This command does not apply to bisynchronous terminals.

CP - CURSOR POSITIONING AFTER INPUT

The CP command determines whether or not the system sends the terminal a response to the line feed key, the EL character, and the EB character.

The format is:

```
ct CP=option
```

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

option Indicates whether to send a response to the terminal or not. A value of Y indicates yes and N indicates no.

This command applies only to asynchronous terminals (except 2741s). If CP=Y and IN=KB are selected, the system responds to the line feed key by sending a carriage return and responds to an EL or EB character by sending the cursor positioning response specified by the EL or EB command. If IN=BK is selected or you are at an X.25 PAD terminal, the system responds only to the EB character by sending the cursor positioning response selected by the EB command.

CT - NETWORK CONTROL CHARACTER

The CT command specifies a network control character for the terminal. When you enter this character as the first character of a logical line, it signals that what follows is a terminal definition command or a special command that the network forwards to your application with a preemptive status (refer to the interactive status commands in the NOS 2 Reference Set, Volume 3).

The format is:

```
ct CT=char
```

ct Represents the current network control character for your terminal.

char Specifies the new network control character. This character must differ from the characters currently defined for AB, BS, B1, B2, CN, EB, and EL.

If you enter the CT command in a line with multiple terminal definitions, the new value char does not take effect until the complete line is processed.

DL - DELIMITERS FOR SINGLE-MESSAGE TRANSPARENT INPUT MODE

The DL command specifies the delimiters that terminate single-message transparent input mode. When you initiate transparent input mode (IN=X, IN=XK, or IN=XP), the network reads your input and sends it to the system without translation until it encounters a delimiter. The first delimiter encountered terminates transparent input mode.

The format is:

```
ct DL=Xxx,Ccount,TO
```

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

Xxx — Specifies the two-digit hexadecimal code (xx) of the character you want as a delimiter. The network does not send the character as part of the input data.

Ccount — Specifies a decimal value (count) from 1 to 4095† that functions as a character count delimiter.

TO — Specifies that a timeout of between 200 and 400 milliseconds will be a transparent mode delimiter.

You can select any of the three types of delimiters: character delimiter (specify Xxx), character count delimiter (specify Ccount), and timeout delimiter (specify TO).

The parameter values for DL are order-independent and optional. However, you must specify at least one of the parameter values. You can omit trailing commas if you do not specify all three types of delimiters.

Entering this command with any number of new delimiters cancels all transparent mode input delimiters already in effect, including those specified with the XL command.

The DL command applies only to asynchronous and mode 4 terminals. Terminal class 4 is configured with the RETURN key as the only transparent mode terminator and mode 4 terminals are configured with only the SEND key as their transparent mode terminator. For paper tape devices, a DC3 following the EL character and, for X.25 devices, the PAD forwarding signal always act as transparent mode terminators in addition to any other options selected.

EB - END-OF-BLOCK CHARACTER

The EB command defines the end-of-block character and defines the cursor-positioning response to that character. The EB character acts as the message transmission key for terminals operating in block mode (refer to the IN command).

The format is:

```
ct EB=eb,cpr
```

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

eb — Specifies the new end-of-block character. You can specify the following values for eb:

eb	Description
xx	Selects the new EB character. This character must differ from the characters currently defined for AB, BS, B1, B2, CN, and CT.
EB	Selects the default end-of-block character.
EL	Selects the current logical end-of-line character.

cpr — Specifies the cursor-positioning response to the EB character. You can specify the following values for cpr:

cpr	Description
CR	Send a carriage return to the terminal.
LF	Send a line feed to the terminal.
CL	Send a carriage return and a line feed to the terminal.
NO	Send no response.

The two EB parameters (EB=eb and cpr) are optional and order-independent. If you omit one, the system retains its previous value.

This command is valid only for asynchronous terminals, except for paper tape devices and 2741 terminals. The end-of-block character for mode 4 terminals is ETX and is not changeable. The X.25 PAD forwarding signal is the default for PSN terminals and always acts as an end-of-block. If you specify the char form of the EB=eb parameter, it is only effective if it occurs as the last character in a packet sequence.

The CP command enables or disables the cursor-positioning response specified by the cpr parameter.

EL - END-OF-LINE CHARACTER

The EL command defines the logical end-of-line character and defines the cursor-positioning response to that character. It also defines the message transmission key for asynchronous terminals not operating in block mode (refer to the IN command).

†These values are valid within the network. However, unless changed during installation, the maximum logical input line length in IAF (160 characters) limits you.

The format is:

ct EL=el, cpr

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

el — Specifies the new logical end-of-line character. You can specify the following values for el:

el	Description
char	Selects the new EL character. This character must differ from the characters currently defined for AB, BS, B1, B2, CN, and CT.
EB	Selects the default end-of-block character.
EL	Selects the current logical end-of-line character.

cpr — Specifies the cursor-positioning response to the EL character. You can specify the following values for cpr:

cpr	Description
CR	Send a carriage return to the terminal.
LF	Send a line feed to the terminal.
CL	Send a carriage return and a line feed to the terminal.
NO	Send no response.

The two EL parameters (EL=el and cpr) are optional and order-independent. If you omit one, the system retains its previous value.

This command does not apply to 2741, HASP, and bisynchronous terminals; the cpr parameter does not apply to mode 4 or X.25 terminals; and the char specification does not apply to mode 4 terminals.

The CP command enables or disables the cursor-positioning response specified by the cpr parameter.

EP - ECHOPLEX MODE†

The EP command enables or disables the echoing of input characters back to the terminal.

The format is:

ct EP=option

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

option — Specifies whether to enable (Y) or disable (N) input echoing.

Some terminals can perform their own character echoing, such as terminals with a HALF/FULL duplex switch. If the switch is in the HALF position, set the EP value to N, and the terminal echoes the input. If the switch is in the FULL position, set the EP value to Y and the network echoes the input.

You can use the EP command and full-duplex mode as a security measure. If you are in full-duplex mode and EP is Y during login, the system echoes all of your input except your password. This is not true, however, if you enter your user name and password on the same line.

This command applies only to asynchronous terminals (except 2741s).

FA - FULL-ASCII INPUT MODE

The FA command enables or disables the full ASCII input mode.

The format is:

ct FA=option

ct — Represents the network control character for your terminal as defined by the CT terminal definition command.

option — Specifies whether to enable (Y) or disable (N) the full ASCII input mode.

In full-ASCII input, all 128 ASCII characters, including NUL, DEL, STX, LF, and all BLK and message-forwarding signals (which are normally discarded), are sent to the host. Exceptions to this are the following characters:

NULs, DELs and LFs when encountered as the first character of an input line or as the character following a message-forwarding signal

CRs that follow LFs when inputting from paper tape

DC3s that follow the EL character when inputting from paper tape

DC1s and DC3s when output control (OC=Y) is selected

In full-ASCII mode, the system does not recognize the terminal functions associated with the mnenomics B1, B2, CN, AB, and BS. The system does recognize terminal definition commands and other commands beginning with the CT character.

If SE is in effect when you select FA or if FA is in effect when you select SE, FA takes precedence and SE will not be effective until FA is turned off. If full-ASCII mode is enabled when you select transparent input mode, the network suspends the full-ASCII mode until the transparent input mode terminates.

†Not valid for PSN terminals.

This command applies only to asynchronous and mode 4 terminals.

HD - DISPLAY OF HOST NODES

The HD command controls when the system displays the host availability display, which shows all paths you can select to connect your terminal to the host.

The format is:

ct HD=option

ct	Represents the network control character for your terminal as defined by the CT command.
option	Represents Y or N. Y enables and N disables the automatic displaying of the HAD.

If you specify HD=Y, you get a display of the HAD immediately and at any time you disconnect from the host (for example, after you enter the TM command or after you log out of an application without switching to another application.)

The HAD has the following format:

HOST	NODE	SELECTED/ CONNECTED	STATUS
host	$node_1$	$condition_1$	$status_1$
host	$node_2$	$condition_2$	$status_2$
.	.	.	.
.	.	.	.
.	.	.	.
host	$node_n$	$condition_n$	$status_n$

The variable items in the display have the following descriptions.

Parameter	Description
host	Represents the 1- to 7-character name of the host.
$node_i$	Represents a node number (1-31).
$condition_i$	Indicates that either you have selected $node_i$ (S), you are connected to the host using $node_i$ (C), you are being auto-connected to the host via $node_i$ (SA), or you have selected and are connected to the host using $node_i$ (SC).
$status_i$	Represents AVAILABLE or NOT AVAILABLE.

HN - HOST NODE SELECTION

The HN command sets or changes the path (node) through which your terminal is connected to the host.

The format is:

ct HN=node

ct	Represents the network control character for your terminal as defined by the CT command.
node	Specifies a node number. The default is the node the network is presently using to access the host.

The host availability display (HAD) lists all nodes your terminal can use. Refer to the HD command for a description of the HAD.

IC - FLOW CONTROL FOR INPUT DEVICES†

The IC command specifies whether the terminal recognizes DC3s as a stop signal and DC1s as a resume signal.

The format is:

ct IC=option

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
option	Specifies whether to enable (Y) or disable (N) input control.

Input control applies to asynchronous terminals, including paper tape devices (except 2741s).

IN - INPUT DEVICE AND TRANSMISSION MODE

The IN command defines the input device and the transmission mode. You can specify the keyboard, block mode keyboard, and the paper tape reader as input devices and you can select transparent input mode (no data conversion on input) or normalized mode for transmission modes.

The format is:

ct IN=in

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
in	Specifies the input device and transmission mode. You can specify the following values for in:

†Not applicable to PSN terminals.

in	Description
BK	Selects block mode as from a form terminal and normalized input mode.
KB	Selects the keyboard and normalized input mode.
PT	Selects paper tape and normalized input mode.
X†	Selects the current input device and transparent input mode.
XK	Selects the keyboard and transparent input mode.
XP	Selects paper tape and transparent input mode.

When the IN value is XK, XP, or X, terminal input is sent to the host in transparent mode blocks. Transparent mode continues until the system encounters a previously defined transparent mode terminator.

When the IN value is XK and transparent mode terminates, the IN value automatically reverts to KB. When the IN value is XP and transparent mode terminates, the IN value automatically reverts to PT. When the IN value is X and transparent mode terminates, the IN value automatically reverts to the value that was previously in effect.

When the IN value is PT or XP, the network sends the X-ON code to start the paper tape reader automatically after the end of a message is output to the terminal and the message empties the terminals's output queue.

The IN command is not applicable to HASP or bisynchronous terminals. 2741 terminals can select only IN=KB, IN=X, and IN=XK.

LI - LINE FEED IDLE COUNT

The LI command specifies the number of idle characters to be inserted into the output stream following a line feed.

The format is:

ct LI=value

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
value	Specifies the new line feed idle count. This value can be from 0 to 99 or the string CA. A value of CA restores the line feed idle count to the default value.

The line feed idle count is similar to the carriage return idle count (CI), except that the idle characters are output after a line feed instead of after the carriage return.

This command applies only to asynchronous terminals.

LK - LOCKOUT OF UNSOLICITED MESSAGES

The LK command determines whether unsolicited messages from the host appear at your terminal.

The format is:

ct LK=option

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
option	Specifies whether to lock out unsolicited messages. A value of Y locks out the messages and a value of N allows the messages to appear.

This command does not apply to bisynchronous terminals.

MS - MESSAGE TO NETWORK OPERATOR

The MS command sends a message to the network operator.

The format is:

ct MS=message

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
message	Specifies your message to the operator. You can use any string of 50 characters or less.

If you are having problems with the network (such as difficulty connecting to IAF), you can communicate these problems to the network operator using the MS command.

OC - FLOW CONTROL FOR OUTPUT DEVICES

The OC command specifies whether the terminal generates DC3s as a stop signal to the network and DC1s as a resume signal.

The format is:

ct OC=option

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.

†PSN and mode 4 terminals, which are always in block mode, can only select IN=X.

option Specifies whether to enable (Y) or disable (N) output control.

If the network receives a DC3, it suspends output until it receives a DC1 or until you send another block of input.

Output control applies to asynchronous terminals, including paper tape devices (except 2741s).

OP - OUTPUT DEVICE SELECTION

The OP command specifies the type of output device.

The format is:

ct OP=op

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

op Specifies the output device. You can specify the following values for op:

op	Description
DI	Specifies a display device.
PR	Specifies a printer.
PT	Specifies a paper tape punch.

You can punch a tape in any mode, but the network does not provide the proper X-OFF characters unless OP has a value of PT.

If you select OP=PR, the network does line folding and page bounding for asynchronous and X.25 terminals.

This command applies to only asynchronous terminals (except 2741s). OP=PT does not apply to X.25 terminals.

PA - PARITY PROCESSING

The PA command specifies the parity processing to be performed by the network.

The format is:

ct PA=pa

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

pa Specifies the type of parity processing. You can specify the following values for pa:

pa	Description
E	Set parity bit for even parity during output, and check for even parity and set parity bit to zero on input.
N	In transparent mode, do not set or check the parity bit because it is part of the input or output data. In normalized mode, set the parity bit to zero on input and output.
O	Set parity bit for odd parity during output, and check for odd parity and set parity bit to zero on input.
Z	Set parity bit to zero on output and check parity bit against zero on input.

This command applies only to asynchronous terminals (except 2741s). For PSN terminals, the parity bit is set to 0 and not checked for input.

PG - PAGE WAITING

The PG command enables or disables page waiting at the terminal during output. If you enable page waiting, the terminal stops at the end of each output page for your acknowledgment (a null input line) before output continues.

The format is:

ct PG=option

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

option Specifies whether to enable (Y) or disable (N) page waiting.

When you enter a null input line to get the next page of output, the null input line has no other meaning.

Page waiting does not apply to bisynchronous or HASP terminals.

PL - PAGE LENGTH

The PL command establishes the maximum number of physical lines that can be printed as one page. If OP has a value of PR, the network automatically positions the carriage to the top of the form after PL lines have been output.

The format is:

ct PL=pl

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

pl Specifies the page length. You can specify 0 or any value from 8 to 255. PL=0 selects an infinitely long page.

The PL value must be nonzero if you want to enable page waiting, since page waiting only occurs after the number of lines defined by pl-1 are output.

PW - PAGE WIDTH

The PW command establishes the maximum number of characters that the terminal prints on one output line. If OP has the value PR, the system automatically starts a new line after pw characters have been output.

The format is:

ct PW=pw

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

pw Specifies the page width. You can specify a value of 0 or any value from 20 to 255. PW=0 selects an infinitely long line.

SE - SPECIAL EDITING MODE†

The SE command enables or disables the special editing variant of normalized input mode.

The format is:

ct SE=option

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

option Enables (Y) or disables (N) special editing operations.

When special editing is in effect, any backspace character entered is stored for transmission to the host. Similarly, a line feed entry produces a normal line feed operation at the terminal and the network stores the line feed character code for transmission to the host. A backspace followed by a line feed causes the system to output an inverted caret (∨) below the last character entered.

This command applies only to asynchronous terminals.

TC - TERMINAL CLASS COMMAND

The TC command specifies a terminal class for your terminal. The terminal class associates your terminal with a predefined set of terminal characteristics.

The format is:

ct TC=tc

ct Represents the network control character for your terminal as defined by the CT terminal definition command.

tc Specifies the terminal class. The value tc can be from 1 to 17.

The following list associates terminal class values with terminal types:

tc	Terminal Type
1	M33, M35, M37, M38 teletypes
2	CDC 713-10, 751-1, 756
3	Reserved for future use
4	IBM 2741
5	M40 teletypes
6	Hazeltine 2000
7	CDC 752
8	Tektronix 4010, 4014
9	HASP Postprint protocol
10	200 User Terminal
11	CDC 714-30
12	CDC 711-10
13	CDC 714-10/20
14	HASP Preprint protocol
15	CDC 734, 731-12, 732-12
16	IBM 2780
17	IBM 3780

A terminal that is not shown as belonging to a terminal class may still be operational. The terminal can be assigned to a terminal class having similar characteristics and the terminal's characteristics can be changed as necessary to correctly define the operational characteristics of the terminal.

The choice of terminal classes you can select is dependent upon the terminal class you were initially assigned. See table F-2 for legal values of TC that can be selected.

If you include a TC command in a sequence of terminal definition commands, enter the TC command first as it resets all other terminal characteristics.

TM - TERMINATING A TERMINAL-HOST CONNECTION

The TM command severs the connection between your terminal and the host.

The format is:

ct TM

ct Represents the network control character for your terminal as defined by the CT command.

†Does not apply to PSN terminals.

You can use the TM command if your application does not respond to your logout attempt. The system responds to the TM command in the following manner:

```
host status    CONTROL CHARACTER IS <ct>
had
prompt
```

The variable items have the following descriptions:

Parameter	Description
host status	Indicates the status of the host you have just specified or to which your are/were connected. You can get any one of the following messages: HOST AVAILABLE HOST BUSY HOST CONNECTED HOST DISCONNECTED HOST UNAVAILABLE NO HOST AVAILABLE NO HOST CONNECTED NO HOST SELECTED
ct	Represents the network control character for your terminal as defined by the CT command.
had	Represents the host availability display. The HAD appears only if you have enabled its automatic display (refer to the HD command).
prompt	Represents the action the system wants you to take. You can get any one of the following prompts: ENTER <CI> HD TO SEE HOST STATUS ENTER <CI> HN=NN TO SELECT HOST ENTER INPUT TO CONNECT TO HOST READY FOR INPUT

XL - MULTIMESSAGE TRANSPARENT MODE

The network provides a transparent mode with a message-forwarding capability that allows you to remain in transparent input mode even after multimessage blocks have been forwarded to the application. The XL command specifies the multimessage-forwarding signal and the terminator for transparent input mode.

The format is:

```
ct XL=Xxx,Ccount,TO,Yyy
```

ct	Represents the network control character for your terminal as defined by the CT terminal definition command.
Xxx	Selects the character with the two-digit hexadecimal representation xx as the message-forwarding signal. If you do not specify a value yy, the occurrence of two successive characters defined by xx acts as a terminator for transparent input mode.
Ccount	Selects a decimal value from 1 to 4095† that functions as a character count, a message-forwarding signal.
TO	Selects timeout as a transparent input mode terminator.
Yyy	Selects the character with the two-digit hexadecimal representation yy as a terminator for transparent input mode when it follows xx.

Yyy, if specified, must follow the Xxx parameter; otherwise, it is ignored. The parameters are otherwise order independent. You must specify at least one of Xxx, Xxx and Yyy, or TO to select a transparent mode terminator. When it immediately follows the character defined by Xxx on input, the character defined by Yyy terminates transparent input mode. The characters defined by Xxx and Yyy can be identical or you can even omit Yyy. In either case, two successive input characters defined by Xxx are required to terminate transparent mode.

Using this command cancels all transparent mode terminators and delimiters defined by the DL command or a previous XL command. You can select multimessage transparent input mode by using the XL command in conjunction with the IN command and selecting IN=X, IN=XK, or IN=XP.

This command does not apply to HASP and bisynchronous terminals.

†These values are valid within the network. However, unless changed during installation, the maximum logical input line length in IAF (160 characters) limits you.

You can transmit input to an application in normalized mode (also called character mode) or transparent mode (also called binary mode). In transparent input mode, the network does not convert any of your input but sends it directly to your application. Characters like LF and BS, which the network normally strips from the input stream, become part of the data sent to your application. Appendix F describes how you delimit and transmit input in transparent mode (refer to the DL and XL commands). The default input mode for all supported terminal classes is normalized mode, however. The remainder of this appendix describes how to delimit and transmit terminal input in normalized mode under the default conditions of the various terminal classes. (Appendix F also describes how you can change these defaults.)

END-OF-PHYSICAL LINE (LINE FEED)

In terminal classes 1 through 8, you can terminate a physical line of input with a line feed key. The network also terminates a physical line of output whenever the page width of a terminal is reached (refer to the PW command in appendix F). The network may respond to the line feed key by advancing the cursor or carriage to the beginning of the next line (refer to the CP command in appendix F).

LOGICAL END-OF-LINE (CARRIAGE RETURN)

You terminate a logical line of input with a carriage return key. A logical line can consist of one or more physical lines but not vice versa. The network may respond to the carriage return by advancing the cursor or carriage to the next line (refer to the EL command in appendix F).

MESSAGE TRANSMISSION

You transmit your input to the network with the message transmission key. For non-blockmode devices, the message transmission key is the carriage return key. Hence, for these terminal classes, logical lines are transmitted separately. When the delimiter for the logical end-of-line and transmission key are different (as with block mode devices, including PSN and mode 4 terminals), the terminal stores logical lines until you press the transmission key. It then sends the single transmission consisting of multiple logical lines to the network. The network sends each logical line separately to your application.

Table G-1 shows the keys used by the various terminal classes to perform the three functions just described.

TABLE G-1. DEFAULT MESSAGE DELIMITER AND TRANSMISSION KEYS

Terminal Class	Function		
	Physical End-of-Line (Line Feed)	Logical End-of-Line (Carriage Return)	Message Transmission Key
1	LINE FEED	RETURN	RETURN
2	↓	RETURN	RETURN †
4	ATTN	RETURN	RETURN
5	NEW LINE	RETURN	RETURN
6	LF	CR	CR
7	LINE FEED	CARRIAGE RETURN	CARRIAGE RETURN
8	LF	RETURN	RETURN
9	Varies††	Varies††	Varies††
10	None	RETURN	SEND
11	None	RETURN	SEND
12	None	NEW LINE	ETX
13	None	NEW LINE	ETX
14	Varies††	Varies††	Varies††
15	None	NEW LINE	SEND
16	None	EM	EOT
17	None	EM	EOT

†If in block mode, refer to terminal documentation for terminal key equivalences. Those listed are for character and line modes.

††Terminals operating under HASP protocol use different keys for this purpose.

This appendix describes how to connect your terminal to the network, to access an application program, and to disconnect a terminal from the network. As used in this appendix, the term application program refers to Control Data Corporation (CDC) programs that provide services to terminal users. You can access application programs from the following list that are installed at your site:

Remote Batch Facility (RBF)

Interactive Facility (IAF)

Transaction Facility (TAF)

Terminal Verification Facility (TVF)

Network Validation Facility (NVF)

Message Control System (MCS)

Other application programs, written at your site, also may be available.

For illustration, this appendix uses RBF to show how a typical application program works within the network. All connection switching on a log shown is true for RBF only.

Before using this appendix, review the applicable terminal operator's manual, which describes the procedures for starting your terminal and the keys for sending information to the host computer.

Each supported terminal has a key the network associates with message transmission. For example, this key can be labelled RETURN, ATTN, SEND, or CR. When you press one of these message transmission keys on a terminal, the network considers that to be the end of the message and sends the message to the host. This appendix uses the carriage return or the boxed CR symbol to refer to a typical key that can be used to send messages from a terminal to the host or to cause a carriage return.

Many different kinds of terminals are available. Supported terminals are divided into two categories: asynchronous and synchronous. Terminals belonging to terminal classes 1-8 are asynchronous and those belonging to terminal classes 9-17 are synchronous.

If a terminal has only an input and an output device that can be used for two-way dialog with the network, the terminal is an interactive console. If the terminal has devices that are used for either input or output only, the terminal is also a batch terminal. Some interactive consoles use a television-like screen or a typewriter-like device to convey information. All interactive consoles have a keyboard for entering information to the network.

Figure H-1 shows a console with a display screen and a keyboard. Figure H-2 shows a console with a printer.

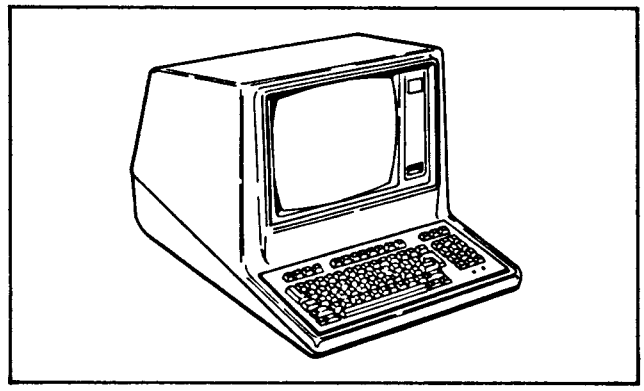


Figure H-1. Interactive Display Console

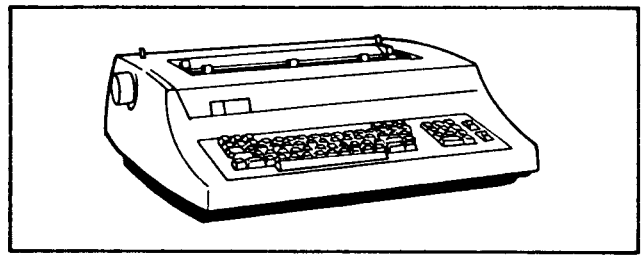


Figure H-2. Interactive Printer Console

SETTING UP YOUR TERMINAL

Before you go further, you should know the following:

The characteristics of the terminal you are using

How the terminal is coupled to the computer

The phone number that corresponds to the class and line speed of the terminal you are using

The family name, user name, and password

Your site administration personnel should be able to give you this information.

Switch settings are important. The switches and their settings vary from terminal to terminal. If other people have used your terminal to connect to the system, the switches should be set correctly; do not change them. Directions for setting the switches may be posted near your terminal.

Here is a procedure to follow when setting up your terminal:

1. Turn on the power switch.

2. Load and initialize any software needed by your terminal. You will find detailed information on this task in the Terminal Interface Guide.

3. Set the on-line/off-line switch to the position that permits on-line communications. For example, on a CDC 713, turn off the LOCAL indicator switch. On a Teletypewriter, set the LINE/OFF/LOCAL switch to the LINE position. On a CDC 200 User Terminal display keyboard, set the ATTENDED/UNATTENDED switch to the ATTENDED position.

4. Set the terminal's duplex or echoplex switch to the correct position (usually half) for proper echoing by your terminal, coupler, or modem. The Terminal Interface Guide describes how to enable and disable character echoing with the network processing unit (NPU). For terminals connected to a packet switching network (PSN), follow the access instructions provided by the PSN.

5. Set the terminal's parity switch to the proper position for the terminal's class. Values for each terminal class are given in the Terminal Interface Guide. (For terminals connected to a packet switching network (PSN), follow PSN access instructions.)

6. Set the line speed switch to a number that matches one of those associated with the phone numbers you were given. If you are not sure of the line speed you should use, set the switch to any position for synchronous terminals (usually 30 characters per second is appropriate for asynchronous terminals). The system may be able to detect automatically the terminal's line speed.

7. Set the transmission mode switch to either the line or character position.

You are now ready to connect your terminal to the system.

CONNECTING YOUR TERMINAL

To begin communication, your terminal must be physically connected to the network. Some terminals, called hardwired, are always connected. Other terminals, called dial-up, are connected to the system using a telephone. If there is not a telephone near the terminal you are using, you can assume its hardwired, and you can skip the remainder of this section; this section applies to dial-up terminals.

A terminal that needs a telephone (dial-up) uses either an acoustic coupler (asynchronous terminals only) or a data set (asynchronous or synchronous terminals) to link itself to the NPU.

An acoustic coupler can be built into the terminal. Figure H-3 shows a terminal with a built-in acoustic coupler and gives directions to connect the terminal to the network.

An acoustic coupler can also be separate from the terminal. Figure H-4 shows an acoustic coupler separate from a terminal and gives directions to connect the terminal to the network.

1. Pick up receiver.
2. Dial phone number.
3. Wait for high-pitched tone.
4. Fit receiver into coupler.

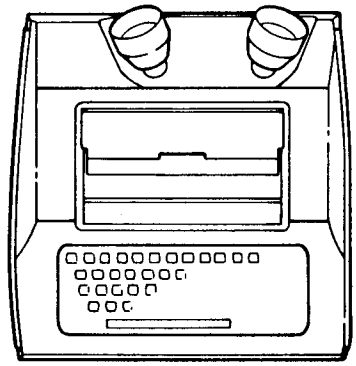


Figure H-3. Terminal With Built-in Acoustic Coupler

1. Turn on coupler.
2. Pick up receiver.
3. Dial phone number.
4. Wait for high-pitched tone.
5. Fit receiver into coupler.

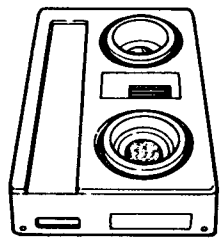


Figure H-4. A Separate Acoustic Coupler

After you dial the phone number, you either get a high-pitched tone or an operator answers. If an operator answers, ask for your terminal to be connected, and wait for the high-pitched tone. If you get a busy signal, wait and redial or try another number.

A data set can be built into the terminal. Figure H-5 shows a terminal with a data set built in and gives directions to connect the terminal to the network.

A stand-alone data set can have either a switch labeled TALK and DATA or individual buttons labeled TALK and DATA. Figure H-6 shows a data set with switches and a data set with buttons, and gives directions to connect the terminal to the network.

An indicator on the terminal (sometimes marked either DSR or Data Set Ready) usually lights to let you know that the terminal is connected to the network. Wait for two seconds after the light comes on before entering data.

IDENTIFYING YOUR TERMINAL TO THE NETWORK

Once you have established physical connection to the NPU, you may need to identify the terminal to the network. This identification procedure, called auto-recognition, is not always needed. For example, it does not apply to PSN-connected terminals. Auto-recognition is necessary if the network must correctly determine the type, character code set, or line speed of the terminal. The network can be configured so that these terminal characteristics are automatically recognized from the first few entries you enter.

If your terminal does not display any information within a few seconds of physical connection, you have one minute to enter one of several identification procedures discussed in the next sections. The version you enter depends on the terminal class. If you do not complete the identification procedure within the allowed time, the network disconnects a dial-up terminal. Hang up the phone and redial the number if you are using a dial-up terminal. You must restart the procedure for a hardwired terminal.

PROCEDURE FOR ASYNCHRONOUS TERMINALS

For an asynchronous terminal, which belongs to terminal classes 1 through 8, use this procedure:

1. Press the carriage return key to identify the line speed used by your terminal. The network software responds with one line feed.
2. Enter a right parenthesis (not required if the terminal is ASCII).
3. Press the carriage return key within one minute to identify the character code set used by the terminal. The network software responds with two line feeds.

1. Pick up receiver.
2. Push button, such as ORIG button on Teletype model 33 or 35.
3. Dial phone number.
4. Wait for high-pitched tone.
5. Push button, such as ORIG button on Teletype model 33 or 35.
6. Replace receiver.

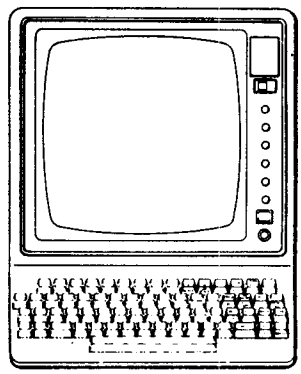
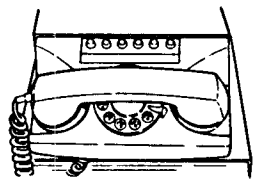


Figure H-5. Data Set Built Into the Terminal

1. Pick up receiver.
2. Set switch to TALK or push TALK button.
3. Dial phone number.
4. Wait for high-pitched tone.
5. Set switch to DATA or push DATA button.
6. Replace receiver.

Data set with switch



Data set with buttons

Figure H-6. Data Sets With Switches and Buttons

PROCEDURE FOR HASP TERMINALS

HASP terminals belong either to terminal class 9 or 14. You do not need to identify the terminal line speed or character code set for HASP terminals. But you must enter a configuration statement that describes the terminal and the devices available.

The format of the configuration statement is:

```
/*CONFIG[,ti,co,dt1,dt2,dt3,dt4]
```

Blanks are not allowed within the statement; a blank ends parameter processing. The parameters, which you may specify in any order, are as follows:

ti is the terminal type indicator:

POST HASP post-print (default); this is terminal class 9

PRE HASP pre-print; this is terminal class 14

co is the configuration ordinal:

CO=xxx A decimal integer from 1 to 255 (default=1)

Use the configuration ordinal (CO) to select one of several terminal definitions defined for the line that describes a particular combination of additional characteristics. Site administration personnel can tell you the correct number to use.

dt is the batch device availability indicator:

dt1 is CR=x - card reader

dt2 is LP=x - line printer

dt3 is CP=y - card punch

dt4 is PL=z - plotter

where x, y, or z can be a list of numbers specified by

1/2/ ... /n

or

ALL - All numbers from 1 to 7

The numbers you use must match the stream numbers used for the corresponding devices within the workstation. Site administration personnel should have this information if you do not know what numbers to use.

y cannot equal z. If you do not specify CR=x, LP=x, or CP=y, 1 is assumed. You cannot omit PL=z if you have a plotter.

PROCEDURE FOR MODE 4 TERMINALS

Mode 4 terminals belong to terminal classes 10 through 13 and 15. Automatic recognition is simple: press the carriage return key within one minute of physical connection.

PROCEDURE FOR BISYNCHRONOUS TERMINALS

You do not need to identify the line speed or character code set for bisynchronous terminals, which belong to terminal classes 16 and 17. Enter a configuration statement to specify the type of terminal and the devices available at it.

The format of the configuration statement is:

```
/*CONFIG[,ti,co,dt1,dt2,dt3]
```

Blanks are not allowed within the statement; a blank ends parameter processing. The parameters, which you may specify in any order, are as follows:

ti is the terminal type indicator:

2780 IBM 2780 (default); this is terminal class 16

3780 IBM 3780; this is terminal class 17

co is the configuration ordinal:

CO=xxx A decimal integer from 1 to 255 (default=1)

Use the configuation ordinal (CO) to select one of several terminal definitions defined for the line that describes a particular combination of additional characteristics. Site personnel can tell you the correct number to use.

dt is the batch device availability indicator:

dt1 is CR

Card reader; assumed to exist if not specified

dt2 is LP

Line printer; assumed to exist if not specified

dt3 is CP=y

Card punch; assumed not to exist if not specified (y is not allowed for 2780; for 3780, y is either 2 or 3 to correspond to the component selection character DC2 or DC3)

PATHS TO A HOST

More than one path may be available to a host from a remote NPU. If one path fails, you can select an alternate route from the remote NPU to the host as shown in figure H-7.

Several Terminal Interface Program (TIP) commands help you to select and use an alternate path to the host. TIP commands are discussed in detail in appendix F of this book.

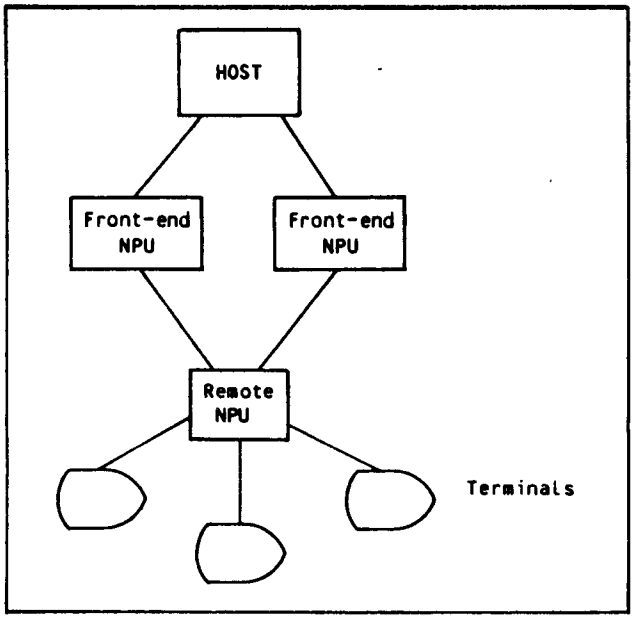


Figure H-7. Alternate Routing to Host

SELECTING A HOST

Your installation personnel can select a path to the host when the network is configured. You can select or change a site-defined path to a host after you connect your terminal to the network.

Use this TIP command to select or to override the NPU routing path to the host:

<ct>HN = nn [CR]

The host node number (1 through 31) is indicated by nn.

Any path you select remains in effect until either you change it or until your terminal is disconnected from the NPU. If you omit the node number, your terminal uses the path the NPU is using to communicate with the host. This is the default.

CONTROLLING THE DISPLAY OF HOST PATHS

You can also turn on and off the automatic displaying of the Host Availability Display (HAD) or request the display with this TIP command:

<ct>HD = {Y / N} [CR]

As shown in figure H-8, the Host Availability Display lists all paths your terminal can currently use to gain access to the host. Initially the display mode is N (no), which means that you receive only two lines at your terminal (user status message and the action message) when either you initially connect to the NPU or logout from the host instead of the full display. After you have entered HD=Y or HD, the full display is issued when either you first connect to an NPU or logout from the host. The display also is issued in response to the HD=Y or to the HD command.

CONNECTING TO THE SELECTED HOST

After you receive a HAD display and select a path to the host, you can ask to be connected to the host by entering a type-in that consists of any input other than a TIP command. If the type-in is not an empty line, any characters you enter are discarded.

If your type-in is rejected, you are notified. If you select another host path you must enter another type-in to accomplish a connection.

Your site personnel may have configured your terminal so that it is auto-connected with the host. The NPU continues to attempt connection indefinitely until the connection is completed. Then the auto-connection is forgotten until either you log out or physically disconnect your terminal or until site personnel are forced to reconfigure the network.

ENDING A HOST CONNECTION

You normally end a connection with a host by using a command to disconnect you from an application or a command to log out of the host. Events might make it impossible for you to do either. When this happens, you can end your terminal's connection to the host by entering this TIP command:

<ct>TM [CR]

The network responds to your TM command with a new Host Availablity Display. You may need to use the TM command if your application program does not respond.

IDENTIFYING YOUR TERMINAL TO THE HOST

The procedure that identifies you and your terminal to the host system is called login. Figure H-9 shows a sample login from an asynchronous terminal. In the example, the terminal has been configured by site personnel for automatic connection to the IAF application program. Everything you would typically enter is shown in lowercase.

Wait two seconds after the light comes on that indicates your terminal is connected to the network. Then press the carriage return twice, using the ASCII default, which allows for a right parenthesis followed by a carriage return. Thereafter, press the carriage return once after each entry. The system responds to each carriage return with one blank line.

You must spell every entry correctly. If you make a mistake, such as typing the letter O for the numeral 0, you will receive an error message. Try again. Generally, you are allowed as many chances as necessary to make a correct entry. However, if you fail to login successfully after four successive attempts, your terminal is automatically disconnected from the host. If this happens, check the user name, spelling, and procedures used. Try again.

```
user status message          CONTROL CHARACTER = ct

HOST          NODE          SELECTED/          STATUS
                            CONNECTED

host1         node1         condition1         status1
host2         node2         condition2         status2
  .             .             .                  .
  .             .             .                  .
  .             .             .                  .
hostn         noden         conditionn         statusn

action message
```

user status message	Status of a given host, which can be one of the following:		You also see:
	HOST CONNECTED	You are connected to the host	action message d or e
	NO HOST SELECTED	Host is available; you have not selected a host	action message b
	NO HOST AVAILABLE	No hosts are available	action message a
	HOST UNAVAILABLE	You have selected a host that is not up	action message a
	HOST BUSY	Connection rejected by host	action message a
	HOST DISCONNECTED	Connection to host terminated	action message c or e
	HOST AVAILABLE	You have selected host that is up	action message c
	NO HOST CONNECTED	You are not connected to a host	action message c

ct	Network control character currently defined for your terminal.
hosti	The 1- to 7-character name of a host computer; if the network only has one host, this can be blank.
nodei	The host node number used in selecting a path through the network; 1 nodei 31.
conditioni	S = selected, not connected C = connected, not selected SC = selected and connected SA = selected, attempting connection
statusi	Either AVAILABLE or NOT AVAILABLE; a host is only available when it is connected to the network.
action message	Action to take in response to a user status message, which can be one of the following: Action message a: ENTER ct HD TO SEE HOST STATUS Action message b: ENTER ct HN=nn TO SELECT HOST Action message c: ENTER INPUT TO CONNECT TO HOST Action message d: READY FOR INPUT Action message e: TERMINAL DISABLED BY NOP

Figure H-8. Host Availability Display

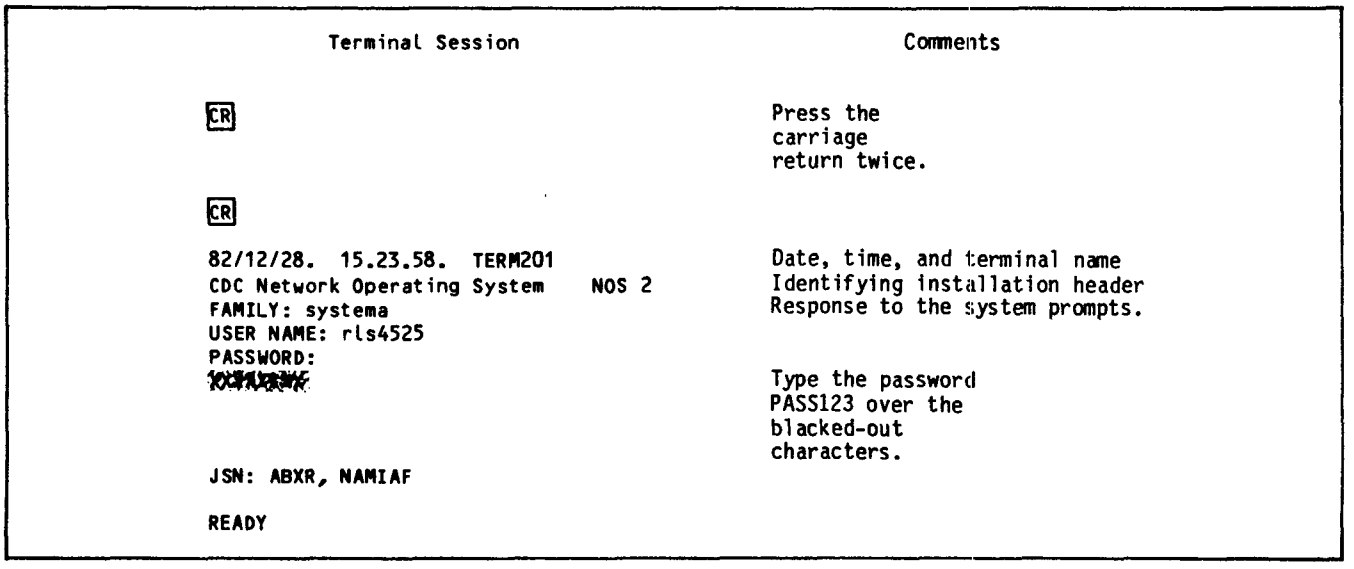


Figure H-9. Sample Login

STANDARD LOGIN

Login begins when the system displays three lines. The first line contains the date, time, and terminal name:

82/12/28. 15.23.58. TERM201

The second line is the identifying header of your computer center:

CDC NETWORK OPERATING SYSTEM NOS 2

The third line is a prompt:

FAMILY:

Enter the 1- to 7-character name of the storage device that contains your permanent files and press the carriage return. If you are using a family name supplied by the system (a default name), just press the carriage return after the FAMILY prompt appears.

The next prompt asks for your user name:

USER NAME:

Enter the user name you were given. The user name, which can contain any combination of digits, letters, and asterisks, identifies you as the terminal operator.

The next system prompt is:

PASSWORD:

or

PASSWORD:
XXXXXXX

As shown in the second password prompt, some terminals, to preserve password secrecy, overprint several characters on a line and ask you to type your password on that line. After creating this row of overstruck characters, the cursor moves back to the first character. Enter the 4- to 7-character password currently associated with your user name. If the network is echoing all input back to the terminal, the network does not echo characters while you are entering your password.

If the family name, user name, and password are acceptable, the next prompt is:

termname - APPLICATION:

The termname variable on this line is the same terminal name as the one on the first line of your login in sequence. Enter the name of the network application program you want your terminal to be connected to; for example, IAF (the Interactive Facility). If your terminal is automatically logged into IAF, you do not receive the APPLICATION prompt.

The application program you asked to be connected to may print an identification line. Login is complete.

ABBREVIATED LOGIN

You can shorten the standard login procedure by entering all of the login information at once. For example, in response to the FAMILY prompt, you can enter the following information on the same line:

FAMILY:
familyname,username,password,application CR

Separate all entries with a comma. The entries are order-dependent. Use a comma to indicate any default or unused entries. You will not receive subsequent prompts if you use this abbreviated log-in procedure and the system cannot do anything to keep your password secret.

Here is an example. To log in and connect to the Remote Batch Facility application program RBF with a family name of systema, a user name of rls4525, and a password of PASS123, enter:

```
systema,rls4525,pass123,rbf [CR]
```

To use the default family name, enter:

```
,rls4525,pass123,rbf [CR]
```

You also can supply combined entries in response to the USER NAME and PASSWORD prompts. Although you do not need to complete the remaining sequence on the same line, you must enter the information in the correct order: family name, user name, password, and application. For example:

```
FAMILY: systema [CR]
USER NAME: rls4525,pass123 [CR]
termname - APPLICATION: iaf [CR]
```

AUTOMATIC LOGIN

A standard system prompts you for the family name, user name, password, and application. Your computer center may elect to preassign family name, user name, or application to any given console. Or, your center may permit you access only to one application through the NOS validation file. If so, you may not see all the login prompts at your terminal and an application program, such as RBF, may be connected automatically.

If you must enter the user name manually from the terminal, you also must enter the password.

Types of Family Names and User Names

An automatic login (preassigned) family name or user name can be either mandatory or default. If mandatory values are preassigned, you will not receive a prompt and will not be able to use any other value. If a default value is preassigned, you have a choice of actions: either enter an empty line as a response to the system prompt if you want either the default family or user name to apply, or respond to the prompt with another value.

Enter a value of 0 (zero) to override a preassigned default family name with the system default family name.

Types of Applications

A preassigned automatic login application may be either mandatory or primary for the terminal or mandatory for the user. If either you or your terminal has a preassigned mandatory application, you will be automatically connected to that application and you will not be able to use any other application.

To override a primary automatic login application on the first attempt to select an application, enter a different application in an abbreviated login input following an earlier prompt. You will be prompted on subsequent attempts to select an application, such as when you are switching applications.

When you are prompted to enter an application name and you want to be connected to your primary application, you may enter an empty line as a response.

LOGIN DIAGNOSTICS

You will receive diagnostic messages if your login was unsuccessful. The following section describes these messages and your responses to them.

OVERRIDEN USER/PASSWORD

If you supply a user/password combination that is overridden by a mandatory specification, this message is displayed:

```
VOLUNTEERED USER/PASSWORD IGNORED.
```

If you offer an application name that is overridden by a preassigned mandatory application, the following message is displayed:

```
VOLUNTEERED APPLICATION IGNORED.
```

UNACCEPTED LOGIN INFORMATION

You will receive the following message if you enter an unacceptable family name, user name, or password:

```
IMPROPER LOGIN, TRY AGAIN.
FAMILY:
```

or

```
IMPROPER LOGIN, TRY AGAIN.
USER NAME:
```

if a mandatory family name is preassigned by your installation.

Reenter all needed login information, beginning with the requested parameter, regardless of when the error occurred.

TOO MANY LOGIN ATTEMPTS

If you make more than four unsuccessful attempts to enter family name, user name, and password, the following message is displayed:

```
USER RETRY LIMIT.
```

Your terminal is disconnected from the host.

ERROR REQUIRES INTERVENTION

If your installation makes an error in preparing your system, one of these messages is displayed at your terminal:

```
USER ACCESS NOT POSSIBLE - CONTACT
NETWORK ADMIN.
```

or

```
LOGIN NOT POSSIBLE - CONTACT NETWORK
ADMIN.
```

You will be disconnected from the host. Ask your site personnel to remedy the problem before you try to access the system again from the same terminal.

UNAVAILABLE APPLICATION PROGRAM

If a mandatory application program is not running in the host, you will be logged out. If you requested an application program that cannot be connected to your terminal because the application is not running in the host, you will receive this message:

```
APPLICATION NOT PRESENT.
termname - APPLICATION:
```

The terminal name is indicated by termname. Either enter the name of another application program or log out.

CONCURRENT CONNECTION TO SAME APPLICATION

Sometimes a system is configured so that only one terminal with a given user name can be logged into the same application program. Although you correctly log in, your request for connection to a particular application will be denied if the same application program is being used by someone else who is logged in with the same family name and user index. You also will be denied a connection if the application is already connected to the maximum number of terminals it can support. In either case, you will receive this message at your terminal:

```
APPLICATION BUSY, TRY AGAIN LATER.
termname - APPLICATION
```

The terminal name is specified by termname. Although you must wait until a current user has finished before you can connect to the same application program, you can request connection to another application program.

UNKNOWN OR UNAVAILABLE APPLICATION

You will receive the following message if the application name you entered in response to the APPLICATION prompt is not valid or if the application is not available to the user name you gave during login:

```
ILLEGAL APPLICATION, TRY AGAIN.
termname - APPLICATION:
```

The terminal name is indicated by termname.

TOO MANY INVALID APPLICATION NAMES

After you have made four unsuccessful attempts to enter an application name, this message appears at your terminal:

```
APPLICATION RETRY LIMIT.
```

Your terminal will be disconnected from the host.

EXCESSIVE RESPONSE TIME

You must enter a response to any login prompt within two minutes. If you take too long to respond to a prompt, this message is displayed at your terminal:

```
TIMEOUT.
```

Your terminal is then disconnected from the host.

HOST DISCONNECTION

Just before your terminal is disconnected from the host, one of two messages is displayed at your terminal:

```
LOGIN TERMINATED.
```

or

```
LOGGED OUT.
```

depending on what caused the disconnection.

SWITCHING TO A DIFFERENT APPLICATION

Sometimes you can transfer your terminal's connection from the current application program to another application by entering a switch command recognized by the current application. For example, if your terminal is connected to RBF, you can gain access to IAF by entering the IAF command. Or, you can switch console connection from RBF to an application program other than IAF by entering the END command. You may receive a prompt for a new application name if you did not specify the next application name on the switch command. Do not enter any more input until the application acknowledges it has processed your switch command. Otherwise, you may lose typed-ahead input.

After disconnecting from the initial application program, the system indicates that a connection switch is in progress by issuing this message:

```
application name - CONNECT TIME hh.mm.ss.
```

The application program that was just disconnected is indicated by application name and the time that the application program was connected to the network is indicated by hh.mm.ss.

After this message, you may receive another prompt for an application program name:

```
termname - APPLICATION:
```

unless the terminal is permitted access to only one application program. If so, you are disconnected from the host.

If you enter the name of another application program in response to the APPLICATION prompt, you are connected to that application program. If you enter an invalid application name, the following message appears:

```
ILLEGAL APPLICATION, TRY AGAIN.
termname - APPLICATION:
```

The terminal name is indicated by termname.

DISCONNECTING FROM THE NETWORK

Three steps always occur when you disconnect from the network. First, you must end the session with the application. Second, you must log out from the host. Third, you must disconnect from the network software.

DISCONNECTING FROM AN APPLICATION

End the session with an application program by entering a disconnection command recognized by that program, such as END. Do not enter any more input until you receive a message acknowledging that your command has been processed. Otherwise, you may lose your input.

You will then receive an APPLICATION prompt.

You can end the connection to RBF and the host by entering either the LOGOUT or LOGOFF command. If you do, you will not receive an APPLICATION prompt.

LOGGING OUT AND DISCONNECTING FROM THE HOST

When you respond to the APPLICATION prompt with the command:

BYE [CR]

or

LOGOUT [CR]

you receive this message at your terminal:

LOGGED OUT.

LOGGING OUT WITHOUT DISCONNECTING FROM THE HOST

If you respond to the APPLICATION prompt with the command:

HELLO [CR]

or

LOGIN [CR]

the terminal is logged out and disassociated from your user name. Login restarts. This allows you to log out without disconnecting your terminal so that another person can use your terminal.

RECONNECTING OR DISCONNECTING FROM THE NETWORK

After your terminal is logged out from the host, this message is printed:

HOST DISCONNECTED.

and followed by the suggested action message:

ENTER INPUT TO CONNECT TO HOST

You can then attempt another connection to the host by sending any input you want. The NPU timesout and physically disconnects dial-up devices if you do not attempt connection within two minutes.

Once a dial-up terminal is disconnected, you can reconnect it only by redialing. The system logoff procedure does not disconnect a hardwired terminal from the network.

CONNECTION FAILURES AND COMMUNICATION INTERRUPTIONS

Some of the problems that can occur while your terminal is connected to the network may cause your application program to fail, your terminal to be disconnected, or may end your terminal's connection to the host.

APPLICATION PROGRAM FAILURES

If an application program fails after your terminal is connected, this message is displayed:

```
APPLICATION FAILED.
application name CONNECT TIME hh.mm.ss.
termname - APPLICATION:
```

The length of time your terminal was connected to the application program is specified by hh.mm.ss; application name is the name of the application program; and termname represents the terminal name the network uses to identify the terminal that is connected to the host. You either may select another application or log out.

INTERRUPTED COMMUNICATIONS

Although you successfully gained access to a network application program, events can sometimes interrupt that access. These interruptions fall into three categories:

Dialog interruptions that convey administrative information

Temporary suspensions in communication

Communication failures that are not immediately recoverable

Administrative Interruptions

Administrative interruptions either occur when communication between the network operator and the terminal is necessary, or when the application program you are using detects an event that causes it to end communication with your terminal. You might be aware of the first type of interruption because the network operator (NOP) may need to communicate with you. Depending on the application program involved, the second type of interruption can occur without your knowledge, except for the subsequent APPLICATION prompt.

Communicating With the Network Operator

The network operator can send any desired text message to a terminal. Any message from the network operator to your terminal has this format:

FROM NOP ... message text

You may see such messages at any time a new line of output is possible at your terminal. For example, the network operator might want to inform all connected terminals that communication is going to be suspended intentionally with this message:

```
FROM NOP ... SHUTDOWN IN 5 MIN
```

You either can respond to such messages, or communicate with the network operator by entering a line with this format:

```
<ct>MS=message text [CR]
```

The message text must not exceed 50 graphic characters. However, because the network operator can be located either at a terminal that does not support many special characters, it is advisable to avoid characters not in the ASCII 63-character subset described in appendix A.

You can also begin dialog with the network operator. For example, if you need to use a remote batch printer that has been logically disabled, the entry:

```
%MS=PLEASE ENABLE LP=M4C555 [CR]
```

causes the message PLEASE ENABLE LP=M4C555 to appear at the network operator's terminal, prefixed by the terminal name of your terminal. The % character is the default network control character for synchronous terminals.

Timing Out and Shutting Down

An application program is informed of two events that you might want to take action on:

- Long periods of terminal inactivity
- Pending network shutdown

The network software informs an application if more than 10 minutes have elapsed without any communication between your terminal and the application. An application program can ignore this time interval, solicit continued dialog from you, or disconnect from your terminal with or without telling you. The effect of terminal inactivity depends on the application program the terminal is currently connected to.

When shutdown of the network is pending, the network operator can enter a command that informs all executing application programs of the pending event. Your application program might take actions that either result in connection termination or application failure. It also might send you a message to warn you of the event.

If you receive a message from an application program indicating that shutdown is pending, you should take steps immediately to save all data or files you are currently using. Then end connection with both the application program and the host as soon as possible.

Suspensions of Communication

Sometimes, message traffic in the network is so heavy that all storage is temporarily used up. Also, a host may be too busy to accept data. During these times, the network software takes steps that temporarily stop input until enough output has occurred to free additional storage.

Although storage may be low, sometimes the network cannot prevent you from entering data. Therefore, the network discards each message from your terminal and sends this message to your terminal:

```
WAIT ..
```

When the following message appears, you may reenter any information that was previously discarded:

```
REPEAT ..
```

If your terminal transmits more than one message at a time, you must determine which information you need to reenter. The application program might have commands that can help you determine which messages it received.

Communication Failures

These messages inform you that the NPU has lost communication with the host software; the user status message:

```
HOST DISCONNECTED.
```

followed by the prompting action message:

```
ENTER INPUT TO CONNECT TO HOST
```

If communication is lost for more than two minutes, dial-up terminals are disconnected from the network.

If communication between the NPU and the host software resumes within two minutes, you must connect to the host and log in again but you do not need to access the network again (perform dial-up and autorecognition procedures). These host communication failures occur either when the host has been shut down or when hardware fails in the communication path between the NPU and the host.

All TIP conditions (that is, parameters that may be set with TIP commands) are retained across host disconnection except for transparent, full ASCII, and special edit input modes. See appendix F for a discussion of these commands.

Failures in the communication path between the NPU and your terminal are treated by the network software as if your terminal had failed. These failures can occur because of hardware problems, or maybe because you entered an input character during output. If you hold the interactive output interrupt

key down for too long, a communication failure can occur. This key is identified in the Terminal Interface Guide for terminals of all classes.

There are several tests for communication failures. If your terminal has a CTS, ON-LINE, or CARRIER indicator, the light goes out when a failure occurs. At other terminals, if a failure has occurred, requesting the host availability display produces no response from the network.

Batch input and ouput devices associated with the failed console terminate activity, and batch device connections are ended. The network software does not distinguish between dial-up and hardwired terminals when processing terminal failures.

Terminal failures require you to complete new access and login procedures to resume communication with the network. Terminal failures always cause disconnection when the terminal is on a dial-up line.

SAMPLE FORTRAN PROGRAM

This appendix contains an annotated listing of sample FORTRAN program RMV3, the debug log file, and statistics file generated when the program is run, and the configuration information used so that the program could be run. In this sample program, RMV3 is used to refer to the name of the FORTRAN program and the name of the batch job that ran it, while RMV2 is used to refer to the application name. This sample program does not attempt to use all possible supervisory message sequences or other features of the Network Access Method interface to the network software.

Application program RMV2 echoes terminal keyboard input back to the terminal and provides some additional dialog. Possible dialogs are described later in this section.

CONFIGURATION REQUIREMENTS

RMV2 is designed only for the servicing of interactive console devices. This program contains no logic to initialize batch device connections or to test the device type of each connection. RMV2 contains no logic requiring it to be configured as a unique identifier application program. For the purposes of this manual, RMV2 is not configured as a privileged application; it is submitted to the operating system and executed as a batch origin job.

RMV2 is completely configured in the local configuration file by the Network Definition Language statement:

```
RMV2:APPL.
```

and terminal operators must log in to it using this application program name.

Terminals accessing RMV2 can be configured with RMV2 as an initial application program if they have a device type of console.

COMMAND PORTION

Program RMV3 was run using the commands shown in figure I-1. The user name appearing on the NOS USER statement has the CUCP bit set in its associated access word.

Although the command portion uses the version of AIP that generates the debugging and statistical files, RMV3 itself does not contain calls to the routines controlling entries in those files. The files are generated for the entire program by default.

PROGRAM PORTION

Figure I-2 shows the program portion of the RMV3 batch job. The comments in the program explain most of the program's logic. The terminal operator dialog supported by RMV2 includes the text exchanges shown in figure I-3. This figure does not illustrate login dialog or dialog after RMV2 is disconnected from the terminal. The former can be inferred from the connection-request information entered for the connection in the debugging log file created by the AIP code after NETON of RMV2. Note that RMV2 responds to most error conditions or problems by shutting down.

PROGRAM OUTPUT

The FORTRAN code in RMV3 produces several entries in file OUTPUT. Figure I-4 shows the debug log file listing produced by the AIP code in RMV3. The message traffic listed in this file can be compared with the program logic documented in figure I-2 to produce a processing flow diagram for the connection involved. Figure I-5 shows the statistical file listing produced by the AIP code in RMV3.

RMV3. USER(APPL1,PASS,FAM1) CHARGE(0059,2934657) FTN5(LO=S/-A)	Job statement.
LDSET(LIB=NETIOD)	Uses debug and statistical file optional code version of AIP.
LGO. REWIND(ZZZZZSN) DLFP(I=0) COPY(ZZZZZSN)	Disposes of local files containing statistical file and debug log file by copying the first one to OUTPUT and executing the postprocessor to completely list the contents of the second one.

Figure I-1. Control Statement Portion of RMV3 Job

```
      PROGRAM RMV3
C
C NAM 1/CCP 3 REFERENCE MANUAL SAMPLE PROGRAM
C ECHOS INTERACTIVE CONSOLE OPERATOR INPUT
C
C NOTE THAT THE DEBUGGING LOG FILE AND STATISTICS FILE LOCAL NAMES
C ARE NOT REQUIRED ON THE PROGRAM STATEMENT GIVEN ABOVE.
C
      IMPLICIT INTEGER(A-Z)
      COMMON /RMCOM/K(20),L,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20),ENDCN
      COMMON /RMCOM/CONEND,FCRST,ACN,ABN(20),SM(20),ABL(20)
      COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20)
C
C NOTE THAT THE TEXT AREAS ARE SEPARATE FOR DATA AND SUPERVISORY
C MESSAGES. THEIR SIZES ARE CHOSEN FOR THE LARGEST EXPECTED SUPERVISORY
C MESSAGE,ARBITRARILY SUPPORTING UP TO 314 CHARACTERS OF DEVICE
C INPUT DATA.
      COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20)
      COMMON /RMCOM/IABN(20),SMHA,SMTA(63)
      EXTERNAL REPREV,CHKSUM
C
C INITIALIZE AND SET CONSTANTS
C
C THIS IS THE CIRCULAR OUTPUT STACK FOR EACH CONNECTION
C
      DATA INSTAK, OUTSTAK/20*0,20*0/
C
C K IS THE APPLICATION BLOCK NUMBER COUNTER
      DATA K/20*1/
C
C THESE ARE NSUP WORD FIELD MASKS
      DATA S/0"02000000000000000000"/
      DATA I/0"04000000000000000000"/
C
C THIS INITIALIZES THE FLOW CONTROL ALGORITHM FOR ALL
C POSSIBLE CONNECTIONS
      DATA ABL,NB,NACN,ABHIBU,STAK/20*0,20*0,20*0,0,20*0/
C
C PACK MASK FOR CHARACTERS THAT COMPRISE OPERATOR END-CONNECTION
C COMMAND FOR NORMAL DISCONNECTION PROCESSING
C WHICH IS THE CAPITALIZED MNEMONIC ENDCN IN 12-BIT BYTES
      DATA ENDCN/0"010501160104010301 16"/
C
C PACK A CONSTANT FOR SUPERVISORY MESSAGE HEADER WORDS
      DATA SMHDR/0"03000000000004000001"/
C
C PACK A CONSTANT HEADER WORD FOR DISPLAY CODED OUTPUT
C OF BLOCK TYPE 2
C NOTE THAT THE NO-FORMAT-EFFECTOR BIT IS NOT SET BECAUSE ALL OUTPUT
C TO THE DEVICE GENERATED BY THE PROGRAM CONTAINS A BLANK
C AS A FORMAT EFFECTOR CHARACTER.
      DATA DSHDR/0"02000000000020000024"/
C
C NOTE THAT ONLY 10 CHARACTERS OF OUTPUT ARE PERMITTED BY
C THE TLC DECLARED,PLUS A ZERO TERMINATOR WORD FOR THE LOGICAL LINE.
C
C PACK A CONSTANT HEADER WORD FOR DISPLAY CODED OUTPUT
C OF BLOCK TYPE 1
C NOTE THAT THE NO-FORMAT-EFFECTOR BIT IS NOT SET BECAUSE ALL OUTPUT
C TO THE DEVICE GENERATED BY THE PROGRAM CONTAINS A BLANK
C AS A FORMAT EFFECTOR CHARACTER.
      DATA DSHDR1/0"01000000000020000024"/
C  AGAIN, ONLY 10 CHARACTERS ARE PERMITTED, PLUS A TERMINATOR WORD.
C
C CREATE MASK FOR UNIT SEPARATOR INSERTION CODE
      DATA US,US1/0"00370000000000000000",0"70370000000000000000"/
C
C SET OUTGOING SUPERVISORY MESSAGE CONSTANTS
      CONEND=NFETCH(0,L"CONEND")
      FCRST=NFETCH(0,L"FCRST")
C
```

Figure I-2. Program Portion of RMV3 (Sheet 1 of 10)

```
C BUILD A BRANCHING TABLE FOR INCOMING ASYNCHRONOUS SUPERVISORY
C MESSAGES (NOTE THAT THIS TABLE IS USED IN A MANNER
C THAT PERMITS EXPANSION)
      SM(1)=NFETCH(0,L"FCACK")
      SM(2)=NFETCH(0,L"CONREQ")
      SM(3)=NFETCH(0,L"FCINIT")
      SM(4)=NFETCH(0,L"FCBRK")
      SM(5)=NFETCH(0,L"INTRUSR")
      SM(6)=NFETCH(0,L"FCINA")
      SM(7)=NFETCH(0,L"CONCB")
      SM(8)=NFETCH(0,L"FCNAK")
      SM(9)=NFETCH(0,L"ERRLGL")
      SM(10)=NFETCH(0,L"HOP")
      SM(11)=NFETCH(0,L"CONEND")
C     SET RESPONSE BIT FOR THE CON/END/N MESSAGE
      SM(11)=SM(11) .OR.O"100"
      SM(12)=NFETCH(0,L"SHUINS")
      SM(13)=999
C
C SET UP REPRIEVAL CODE TO SALVAGE DEBUG AND STATISTICAL FILES
      CALL RECOVR(REPREV,O"277",LOCF(CHKSUM))
C
C ESTABLISH ACCESS TO THE NETWORK AND BEGIN DEBUG LOG
C FILE CREATION
      CALL NETON("RMV2",NSUP,NSTAT,1,20)
C
C TEST FOR ACCESS COMPLETION
      IF (NSTAT.NE.0) THEN
       PRINT 100, NSTAT
  100  FORMAT (' NSTAT = ',020)
       STOP 111
      ENDIF
C
C     UPDATE NSUP FLAGS,THEN
C PERFORM CONNECTION ESTABLISHMENT PROCESSING AND
C APPROPRIATELY DISPOSE OF OTHER SUPERVISORY
C MESSAGES RECEIVED
   15 CALL NETWAIT(5,0)
      SHUTDWN=0
      CALL LOOKSM (SHUTDWN)
      GO TO (9,15,40,40,40,9,9,9,9,9,9,554,777) ,L
C
C INITIALIZE CONNECTION BY SENDING OUTPUT
   40 HA=DSHDR
      CALL NSTORE(HA,L"ABHADR",ACN)
      TA(1)=" INPUT PLS"
      TA(2)=0
      SEND=1
   39 CALL OUTPT (SEND)
      IF (SEND .EQ. 0) GO TO 38
      IF (STAK(ACN) .EQ. 1) THEN
         SEND=0
         GO TO 39
      ELSE
         GO TO 9
      ENDIF
C
C PAUSE TO ALLOW OUTPUT QUEUE TO CLEAR
   38 CALL NETWAIT(5,1)
      SHUTDWN=0
      CALL LOOKSM (SHUTDWN)
      GO TO (37,38,40,40,38,9,15,9,9,9,15,554,38) ,L
   37 SEND=0
      CALL OUTPT (SEND)
C
C PAUSE FOR INPUT DATA OR A SUPERVISORY MESSAGE
    9 CALL NETWAIT(4095,0)
C
C TEST FOR QUEUED MESSAGES OR DATA BLOCKS
  777 IF((NSUP.AND.I).EQ.0) GO TO 15
C
```

Figure I-2. Program Portion of RMV3 (Sheet 2 of 10)

```
C FETCH QUEUED INPUT FROM A DEVICE
      ALN=1
      CALL NETGETL(ALN,HA,TA,10)
  778 ABT=NFETCH(HA,L"ABHABT")
      ACT=NFETCH(HA,L"ABHACT")
      ACN=NFETCH(HA,L"ABHADR")
      ABHXPT=NFETCH(HA,L"ABHXPT")
      ABHTRU=NFETCH(HA,L"ABHTRU")
      ABHBRK=NFETCH(HA,L"ABHBRK")
      ABHCAN=NFETCH(HA,L"ABHCAN")
      ABHIBU=NFETCH(HA,L"ABHIBU")
      TLC=NFETCH(HA,L"ABHTLC")
C
C BRANCH TO PROCESS DATA BLOCK OR SYNCHRONOUS SUPERVISORY MESSAGE
      IF (ABT.EQ.3) THEN
       PRINT 101,ACN,HA,TA
  101  FORMAT (' SM RECEIVED WHEN INPUT DATA WAS EXPECTED ON ACN = ',I3,
     * //' HA = ',020,/' TA = ',/10(1X,020/))
       GO TO 9
      ENDIF
C
C MAKE ANOTHER ATTEMPT TO FETCH QUEUED BLOCK
      IF (ABT.EQ.0.AND.ABHIBU.EQ.1) CALL NETGET(ACN,HA,TA,63)
      IF (ABT.EQ.0.AND.ABHIBU.EQ.1) GO TO 778
      IF (ABT.EQ.0.AND.ABHIBU.NE.1) GO TO 15
C
C TEST FOR THROW-AWAY INPUT
      IF(ABHCAN.EQ.1) GO TO 40
C
C TEST FOR TYPE-IN OF ENDCN COMMAND
      IF(TA(1).EQ.ENDCN ) GO TO 666
C
C CREATE HEADER WORD TO ECHO INPUT AS OUTPUT
      HA =(HA .AND. O"777777777777774007777") + O"1"
C
C CHANGE APPLICATION BLOCK TYPE TO 1
      CALL NSTORE(HA,L"ABHABT",1)
C
C INHIBIT FIRST CHARACTER AS A FORMAT EFFECTOR
      CALL NSTORE(HA,L"ABHNFE",1)
C
C ECHO INPUT AS OUTPUT, AFTER ADDING A US TERMINATOR
      FULWD=TLC/5
      FWP1=FULWD+1
      XTRA=12*(TLC - 5*FULWD)
      TLC=TLC + 1
      CALL NSTORE(HA,L"ABHTLC",TLC)
      IF (XTRA.EQ.0) THEN
       TA(FWP1)=US
      ELSE
       XXX=SHIFT(US1,-XTRA)
       YYY=SHIFT(US,-XTRA)
C
C ZERO OUT REMAINDER OF WORD AND ADD UNIT SEPARATOR  CHARACTER TO END OF BLOCK
       TA(FWP1)=TA(FWP1) .AND. XXX .OR. YYY
      ENDIF
      SEND=1
      CALL OUTPT (SEND)
C
C SEND ANOTHER PROMPTING LINE IF OUTPUT IS POSSIBLE AND END OF CURRENT
C MESSAGE HAS BEEN RECEIVED.  IF OUTPUT IS NOT POSSIBLE, STALL.
C IF END OF CURRENT MESSAGE HAS NOT BEEN RECEIVED, FETCH IT.
      IF (SEND .EQ. 0) GO TO 38
      IF (STAK(ACN) .EQ. 1) GO TO 38
      IF (ABT .EQ. 1) GO TO 9
      GO TO 40
C
C CLEANUP ALL CONNECTIONS BEFORE ENDING NETWORK ACCESS
  666 SMTA(1)=SMTA(2)=0
      CALL NSTORE(SMTA,L"PFCSFC",CONEND)
      CALL NSTORE(SMTA,L"RC",0)
```

Figure I-2. Program Portion of RMV3 (Sheet 3 of 10)

```
      C
      C PASS CONNECTION DIRECTLY TO IAF WITHOUT DIALOG
            CALL NSTORE(SMTA,L"CONANM",R"IAF    ")
            SMHA=SMHDR + 0"1"
            DO 555 J=1,20
            IF (NACN(J).EQ.1) THEN
             CALL NSTORE (SMTA,L"CONACN",J)
             NACN(J)=0
             CALL NETPUT (SMHA,SMTA)
            ENDIF
        555 CONTINUE
      C
      C FETCH ALL QUEUED SUPERVISORY MESSAGES TO AVOID AN APPLICATION
      C FAILED MESSAGE TO THE DEVICE OPERATOR AFTER DISCONNECTION
         97 CALL NETWAIT(5,0)
            SHUTDWN=1
            CALL LOOKSM (SHUTDWN)
            IF (L.EQ.3) GO TO 666
            IF (L.LE.12) GO TO 97
      C
      C FINISH WRITING DEBUG LOG FILE AND STATISTICAL FILE
        554 CALL NETOFF
            END


      C
            SUBROUTINE LOOKSM (SHUTDWN)
      C
      C PROCESS INCOMING ASYNCHRONOUS SUPERVISORY MESSAGES
      C
            IMPLICIT INTEGER (A-Z)
            COMMON /RMCOM/K(20),L,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20),ENDCN
            COMMON /RMCOM/CONEND,FCRST,ACN,ABN(20),SM(20),ABL(20)
            COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20)
            COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20)
            COMMON /RMCOM/IABN(20),SMHA,SMTA(63)
      C
      C SET MESSAGE BRANCH VALUE FOR RETURN
            L=13
      C
      C FETCH AN ASYNCHRONOUS SUPERVISORY MESSAGE OR WAIT FOR ONE
          3 IF ((NSUP.AND.S).EQ.0) THEN
               IF (SHUTDWN .EQ. 0) CALL NETWAIT (4095,0)
      C
      C RETURN TO FETCH INPUT DATA
             RETURN
            ENDIF
      C
      C FETCH AN ASYNCHRONOUS SUPERVISORY MESSAGE FROM ACN=0
      C ON LIST ZERO
            ALN=0
            CALL NETGETL(ALN,SMHA,SMTA,63)
      C
      C UNPACK THE MESSAGE IDENTIFICATION AND BRANCH ON THE TYPE
            PFCSFC=NFETCH(SMTA,L"PFCSFC")
            PFC=NFETCH(SMTA,L"PFC")
      C
      C NOTE THAT THIS LOOP EXITS WITH THE L VALUE SET SO THAT IT CAN BE
      C USED FOR BRANCHING IN THE MAIN PROGRAM ON RETURN FROM LOOKSM
            DO 8 L=1,20
            IF(PFCSFC.EQ.SM(L))GOTO(10,20,30,40,50,60,70,80,90,100,110,120),L
            IF (SM(L).EQ.999) THEN
      C
      C ISSUE DIAGNOSTIC MESSAGE TO OUTPUT FILE
             PRINT 1000, SMHA,SMTA
       1000  FORMAT (' COULD NOT FIND SM IN TABLE OF SUPPORTED CODES',
           * //' HA = ',O20,/' TA = ',/63(1X,O20/))
             GO TO 3
            ENDIF
            IF (PFC.EQ.SM(L))GOTO(10,20,30,40,50,60,70,80,90,100,110,120),L
      C
```

Figure I-2. Program Portion of RMV3 (Sheet 4 of 10)

```
 47              8 CONTINUE
 48          C
 49          C TRY AGAIN
 50                GO TO 3
 51          C
 52          C PROCESS FC/ACK/R SUPERVISORY MESSAGE
 53             10 ACN=NFETCH(SMTA,L"FCACN")
 54                 IABN(ACN)=NFETCH(SMTA,L"FCABN")
 55          C
 56          C UPDATE FLOW CONTROL ALGORITHM
 57                NB(ACN)=NB(ACN) - 1
 58                RETURN
 59          C
 60          C PROCESS CON/REQ/R SUPERVISORY MESSAGE
 61          C
 62          C UNPACK MESSAGE AND USE CONTENTS TO SET UP CONNECTION
 63          C FLOW CONTROL ALGORITHM
 64             20 ACN=NFETCH(SMTA,L"CONACN")
 65                ABL(ACN)=NFETCH(SMTA,L"CONABL")
 66                NB(ACN)=0
 67          C
 68          C SET RESPONSE BIT TO ACCEPT CONNECTION
 69                CALL NSTORE(SMTA,L"RB",1)
 70          C
 71          C INPUT MUST BE ASCII IN 12-BIT BYTES
 72                CALL NSTORE(SMTA,L"CONACT",3)
 73          C
 74          C ASSIGN ALL INTERACTIVE CONSOLES TO LIST 1
 75                CALL NSTORE(SMTA,L"CONALN",1)
 76          C
 77          C PACK CONNECTION-ACCEPTED PORTION OF MESSAGE
 78                SMTA(1)=SMTA(1).AND.O"777774007777000017777"
 79                SMHA=SMHDR
 80          C
 81          C SEND THE CONNECTION-ACCEPTED SUPERVISORY MESSAGE
 82                CALL NETPUT(SMHA,SMTA)
 83                RETURN
 84          C
 85          C PROCESS FC/INIT/R SUPERVISORY MESSAGE
 86          C
 87          C SET THE RESPONSE BIT TO INDICATE READY FOR
 88          C TRANSMISSION TO BEGIN
 89             30 CALL NSTORE(SMTA,L"RB",1)
 90          C
 91          C DETERMINE LOGICAL CONNECTION INVOLVED AND UPDATE
 92          C CONNECTION TABLE
 93                ACN=NFETCH(SMTA,L"FCACN")
 94                NACN(ACN)=1
 95                SMHA=SMHDR
 96          C
 97                IABN(ACN)=ABN(ACN)=0
 98          C SEND THE CONNECTION-INITIALIZED MESSAGE
 99                CALL NETPUT(SMHA,SMTA)
100                RETURN
101          C
102          C PROCESS FC/BRK/R SUPERVISORY MESSAGE
103          C
104          C UNPACK MESSAGE FIELDS
105             40 RC=NFETCH(SMTA,L"RC")
106                ACN=NFETCH(SMTA,L"FCACN")
107          C
108          C UPDATE FLOW CONTROL ALGORITHM FOR THIS CONNECTION
109          C TO REFLECT THAT ALL DOWNLINE BLOCKS HAVE BEEN DISCARDED
110          C
111                NB(ACN)=0
112                INSTAK(ACN)=OUTSTAK(ACN)=STAK(ACN)=0
113                SMHA=SMHDR
114          C
115          C
116          C PACK RESET SUPERVISORY MESSAGE
117                SMTA(1)=0
```

Figure I-2. Program Portion of RMV3 (Sheet 5 of 10)

```
118                CALL NSTORE(SMTA,L"PFCSFC",FCRST)
119                CALL NSTORE(SMTA,L"FCACN",ACN)
120           C
121           C SEND RESET MESSAGE
122                CALL NETPUT(SMHA,SMTA)
123           C
124           C OUTPUT USER BREAK INDICATION TO THE DEVICE OPERATOR
125                HA=DSHDR1
126                IF (RC.EQ.1) TA(1)=" BREAK 1  "
127                IF (RC.EQ.2) TA(1)=" BREAK 2  "
128                IF ((RC.NE.1).AND.(RC.NE.2)) TA(1)=" BREAK!   "
129                TA(2)=0
130                CALL NSTORE(HA,L"ABHADR",ACN)
131                INSTAK(ACN)=OUTSTAK(ACN)=STAK(ACN)=0
132                SEND=1
133                CALL OUTPT (SEND)
134                RETURN
135           C
136           C PROCESS INTR/USR/R SUPERVISORY MESSAGE
137              50 ACN=NFETCH(SMTA,L"INTRACN")
138                 INTRCHR=NFETCH(SMTA,L"INTRACN")
139                 HA=DSHDR1
140                 TA(1)=" BYPASSED "
141                 TA(2)=0
142                 CALL NSTORE(HA,L"ABHADR",ACN)
143                 SEND=1
144                 CALL OUTPT (SEND)
145                 RETURN
146           C
147           C PROCESS HOST OPERATOR COMMANDS
148             100 CONTINUE
149                 RETURN
150           C
151           C PROCESS FC/INACT/R SUPERVISORY MESSAGE
152           C
153           C UPDATE CONNECTION TABLE
154              60 ACN=NFETCH(SMTA,L"FCACN")
155                 NACN(ACN) = 0
156                 HA=DSHDR
157                 CALL NSTORE(HA,L"ABHADR",ACN)
158           C
159           C OUTPUT DISCONNECTION INDICATOR TO POSSIBLE OPERATOR
160                TA(1)=" TIME OUT "
161                TA(2)=0
162           C
163           C NOTE THAT RMV2 DOES NOT WAIT FOR AN FC/ACK/R CORRESPONDING TO
164           C THIS OUTPUT MESSAGE. AN ERR/LGL/R MESSAGE WILL EVENTUALLY
165           C BE CAUSED BY THE CONNECTION TERMINATION PROCESSING CODE,
166           C CAUSING RMV2 TO NETOFF WITHOUT DEVICE OPERATOR
167           C OR HOST OR NETWORK OPERATOR ACTION BEING REQUIRED.
168                INSTAK(ACN)=OUTSTAK(ACN)=STAK(ACN)=0
169                SEND=1
170                CALL OUTPT (SEND)
171           C
172           C PACK AND SEND CONNECTION-END REQUEST MESSAGE
173           C
174                SMTA(1)=0
175                CALL NSTORE(SMTA,L"PFCSFC",CONEND)
176                CALL NSTORE(SMTA,L"CONACN",ACN)
177                SMTA(2)=0
178                SMHA=SMHDR
179                CALL NETPUT (SMHA,SMTA)
180                RETURN
181           C
182           C PROCESS CON/CB/R SUPERVISORY MESSAGE
183              70 ACN=NFETCH(SMTA,L"CONACN")
184                 PRINT 75,ACN
185              75 FORMAT(' CONNECTION BROKEN, ACN = ',I3)
186           C
187           C FETCH ALL OUTSTANDING INPUT BLOCKS UNTIL A NULL
188           C BLOCK IS RECEIVED
```

Figure I-2. Program Portion of RMV3 (Sheet 6 of 10)

```
      71 CALL NETGET(ACN,HA,TA,63)
         IF (NFETCH(HA,L"ABHABT").EQ.0) GO TO 72
C        DETERMINE WHETHER THIS IS A NORMAL ENDCN SEQUENCE FETCHED OUT OF
C        SYNCHRONIZATION.
C        IF SO, USE THE ERR/LGL/R LOGIC TO SHUT DOWN.
         IF(TA(1).EQ.ENDCN) GO TO 76
         GO TO 71
C
C CLEAN UP CONNECTION TABLE ENTRY AND AIP TABLES
      72 CALL NSTORE(SMTA,L"CONACN",ACN)
         CALL NSTORE(SMTA,L"RC",0)
         CALL NSTORE(SMTA,L"PFCSFC",CONEND)
         SMHA=SMHDR
         NACN(ACN)=0
         CALL NETPUT(SMHA,SMTA)
         RETURN
C
C PROCESS FC/NAK/R SUPERVISORY MESSAGE
      80 ACN=NFETCH(SMTA,L"FCACN")
         ABN(ACN)=NFETCH(SMTA,L"FCABN")
         PRINT 1015,ACN,ABN(ACN)
    1015 FORMAT(' ACN = ',I6,' ABN = ',I10,' NOT DELIVERED')
         RETURN
C
C PROCESS CON/END/N SUPERVISORY MESSAGE
C PROCESSING TREATS THE MESSAGE AS ADVISORY IN ALL CASES.
     110 RETURN
C
C
C PROCESS ERR/LGL/R SUPERVISORY MESSAGE,
C WRITE MESSAGE TO OUTPUT FILE FOR ANALYSIS, THEN SHUT
C DOWN OPERATIONS
      90 PRINT 1000,SMHA,SMTA
      76 SMTA(1)=SMTA(2)=0
         CALL NSTORE(SMTA,L"PFCSFC",CONEND)
         CALL NSTORE(SMTA,L"RC",0)
         SMHA=SMHDR
         DO 333 II=1,20,1
         IF (NACN(II).EQ.1) THEN
          CALL NSTORE (SMTA,L"CONACN",II)
         CALL NETPUT(SMHA,SMTA)
C
C UPDATE CONNECTION TABLE
          NACN(II)=0
         ENDIF
     333 CONTINUE
         CALL NETOFF
         STOP
C
C PROCESS SHUT/INSD/R SUPERVISORY MESSAGE,
C WRITE MESSAGE TO OUTPUT FILE FOR ANALYSIS, THEN
C SHUTDOWN OPERATIONS
     120 PRINT 1000,SMHA,SMTA
         SMTA(1)=SMTA(2)=0
C
C DETERMINE TYPE OF SHUTDOWN
         IBIT=NFETCH(SMTA,L"SHUTF")
         IF (IBIT.EQ.1) THEN
          CALL NETOFF
          RETURN
         ENDIF
C
C SHUTDOWN GRACEFULLY IF TIME PERMITS
         CALL NSTORE(SMTA,L"PFCSFC",CONEND)
         CALL NSTORE(SMTA,L"RC",0)
         SMHA=SMHDR
         DO 444 JJ=1,20,1
         IF (NACN(JJ).EQ.1) THEN
          CALL NSTORE (SMTA,L"CONACN",JJ)
          CALL NETPUT (SMHA,SMTA)
C
```

Figure I-2. Program Portion of RMV3 (Sheet 7 of 10)

```
260           C UPDATE CONNECTION TABLE
261                 NACN(JJ)=0
262                 ENDIF
263             444 CONTINUE
264           C
265           C FINISH WRITING DEBUG LOG FILE
266                 CALL NETOFF
267                 END


  1                 SUBROUTINE OUTPT (SEND)
  2           C
  3           C OUTPUT ONE DATA BLOCK
  4           C
  5                 IMPLICIT INTEGER (A-Z)
  6                 COMMON /RMCOM/K(20),L,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20),ENDCN
  7                 COMMON /RMCOM/CONEND,FCRST,ACN,ABN(20),SM(20),ABL(20)
  8                 COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20)
  9                 COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20)
 10                 COMMON /RMCOM/IABN(20),SMHA,SMTA(63)
 11           C
 12           C IS THERE DATA IN THE MAIN OUTPUT BUFFER?
 13           C
 14                 IF (SEND.EQ.1) THEN
 15           C
 16           C        IF SO, IS THERE SOMETHING ELSE TO SEND FIRST?
 17           C
 18                    IF (STAK(ACN) .EQ. 1) THEN
 19           C
 20           C           IF SO, ADD NEW OUTPUT TO STACK
 21           C
 22                       GO TO 1
 23                    ELSE
 24           C
 25           C        IF NOT, TEST IF NEW OUTPUT CAN BE SENT
 26           C
 27                       GO TO 9
 28                    ENDIF
 29                 ELSE
 30           C
 31           C IF NOT, TEST IF DATA NEEDS TO BE SENT FROM THE STACK
 32           C
 33                    GO TO 8
 34                 ENDIF
 35           C
 36           C IS THERE DATA IN THE STACK?
 37           C
 38               8 IF (STAK(ACN) .EQ. 0) THEN
 39           C
 40           C        IF NOT, EXIT
 41           C
 42                    RETURN
 43                 ELSE
 44           C
 45           C        IF SO, TEST IF IT CAN BE SENT
 46           C
 47                    GO TO 3
 48                 ENDIF
 49           C
 50           C CAN DATA BE SENT?
 51           C
 52               9 IF (NB(ACN) .GE. ABL(ACN)) THEN
 53           C
 54           C         IF NOT, STACK IT
 55           C
 56                    STAK(ACN)=OUTSTAK(ACN)=INSTAK(ACN)=1
 57                    OVRFLHA(INSTAK(ACN),ACN)=HA
 58                    DO 888 JJ=1, 63, 1
 59             888    OVRFLTA(JJ,INSTAK(ACN),ACN)=TA(JJ)
 60                    RETURN
 61           C
```

Figure I-2. Program Portion of RMV3 (Sheet 8 of 10)

```
      C         IF SO, DO IT
      C
           ELSE
      C
      C         UPDATE FLOW CONTROL ALGORITHM
      C
                ABN(ACN)=ACN*64 + K(ACN)
                K(ACN)=K(ACN) + 1
                NB(ACN)=NB(ACN) + 1
                CALL NSTORE(HA,L"ABHABN",ABN(ACN))
                CALL NETPUT(HA,TA)
                RETURN
           ENDIF
      C
      C IS THERE ROOM FOR MORE DATA IN THE STACK?
      C
      C     IF NOT, THROW AWAY NEW OUTPUT
      C
          1 IF (INSTAK(ACN).GT.OUTSTAK(ACN)) THEN
                IF ((INSTAK(ACN) - OUTSTAK(ACN)) .EQ. 7) THEN
                    SEND=0
                    RETURN
                ENDIF
           ELSE
                IF ((OUTSTAK(ACN) - INSTAK(ACN)) .EQ. 1) THEN
                    SEND=0
                    RETURN
                ENDIF
           ENDIF
      C
      C     IF SO, SAVE THE NEW DATA
      C
           INSTAK(ACN)=INSTAK(ACN) + 1
           IF (INSTAK(ACN) .EQ. 9) INSTAK(ACN)=1
           OVRFLHA(INSTAK(ACN),ACN)=HA
           DO 999 II=1, 63, 1
        999 OVRFLTA(II,INSTAK(ACN),ACN)=TA(II)
      C
      C PROCESS DATA ALREADY IN STACK
      C
      C
      C CAN DATA BE SENT?
      C
          3 IF (NB(ACN) .GE. ABL(ACN)) THEN
      C
      C         IF NOT, EXIT
      C
                RETURN
      C
      C         IF SO, DO IT
      C
           ELSE
      C
      C         UPDATE FLOW CONTROL ALGORITHM
      C
                ABN(ACN)=ACN*64 + K(ACN)
                K(ACN)=K(ACN) + 1
                NB(ACN)=NB(ACN) + 1
                CALL NSTORE(OVRFLHA(OUTSTAK(ACN),ACN),L"ABHABN",ABN(ACN))
                CALL NETPUT(OVRFLHA(OUTSTAK(ACN),ACN),
           +       OVRFLTA(1,OUTSTAK(ACN),ACN))
      C
      C         TEST IF STACK HAS BEEN EMPTIED
      C
                IF (OUTSTAK(ACN).EQ.INSTAK(ACN)) THEN
                    STAK(ACN)=0
      C
      C             IF SO, REINITIALIZE POINTERS
      C
                    OUTSTAK(ACN)=INSTAK(ACN)=0
                ELSE
```

Figure I-2. Program Portion of RMV3 (Sheet 9 of 10)

```
133           C
134           C             IF NOT, MOVE THE SEND BUFFER POINTER FOR NEXT PASS
135           C
136                         OUTSTAK(ACN)=OUTSTAK(ACN) + 1
137                         IF (OUTSTAK(ACN) .EQ. 9) OUTSTAK(ACN)=1
138                      ENDIF
139                   ENDIF
140                   RETURN
141                   END


  1                   SUBROUTINE REPREV (IXCHNG,IFLAG,IFLDLN)
  2           C
  3           C THIS SUBROUTINE SALVAGES THE DEBUG AND STATISTICAL FILE ENTRIES BY
  4           C CALLING THE AIP ROUTINE NETOFF TO FLUSH BUFFERS IN CASE THE
  5           C APPLICATION PROGRAM IS ABORTED DURING EXECUTION
  6           C
  7                   DIMENSION IXCHNG(17),IFLDLN(O"30000")
  8                   IFLAG=1
  9                   CALL NETOFF
 10                   STOP
 11                   ENTRY CHKSUM
 12                   END
```

Figure I-2. Program Portion of RMV3 (Sheet 10 of 10)

INPUT PLS	Prompt to operator from RMV2 for first input.
User-break-1 or user-break-2	Entered by terminal operator.
BREAK n	RMV2 response to break entries.
INPUT PLS	Prompt for next input.
BYPASSED	RMV2 response to INTR/USR/R supervisory message.
TIME OUT	RMV2 output documenting an inactive connection; this is followed by disconnection from RMV2 for subsequent terminal operator dialog with NVF or disconnection from the network.
INPUT PLS	RMV2 prompt for next input.
ENDCN	Terminal operator entry, causes normal connection termination for this terminal and for all other connected terminals. Next terminal operator dialog is with IAF, if that program is available.
INPUT PLS	RMV2 prompt for input.
Any characters other than ENDCN, up to 314	Terminal operator entry.
Any characters other than ENDCN, up to 314	RMV2 echoed output, single-spaced.
INPUT PLS	RMV2 prompt for next entry.

Figure I-3. Possible Dialogs Supported by Sample FORTRAN Program

```
10.35.16.000        NETON   (024726)     ANAME = RMV2          DATE = 82/11/16            MSG NO. 000001
 NSUP ADDR = 000140  MINACN =00001  MAXACN =00020


10.36.16.599        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000002
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0011
     001 6300000016002O0 3060000000013O001000  CONREQ       C
     002 51C79D7EDB45018 2434363537555505003O  T1324  E X    YWM4P
     003 0000000000006EA 00000000000000003352          0)        N
     004 00000000000DD409 0000000000000335201I        C2PI       T
     005 [illegible]
     006 [illegible]
     007 [illegible]
     008 [illegible]
     009 [illegible]
     010 [illegible]
     011 [illegible]


10.36.16.604        NETPUT  (030612)   HA =024534   TA =024535                          MSG NO. 000003
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 6340000010000C1 3064000000010000O301  CONREQN      C@


10.36.17.187        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000004
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8307000010000O0 4060340000010000000O  FCINIT


10.36.17.192        NETPUT  (030612)   HA =024534   TA =024535                          MSG NO. 000005
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8347000010000O0 4064340000010000000O  FCINITN       G


10.36.17.193        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000006
 ABT =02  ADR =0001  ABN =000065  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 55111620252455201423    INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0


10.36.17.714        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000007
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010010O0 4060100000010001O100  FCACK


10.36.28.571        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000008
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0020
     001 055073065072O2D 0125016301450162005S  AUA%A+A]    USER-
     002 062072065061O6B 0142016201450141015S  A7A]A+A6A$  BREAK
     003 02D03102006606F 0055006100400146015S      [ 5A-A.  -1 FO
     004 06C06C06F077073 0154015401570167016S  A=A=A.A&A%  LLOWS


10.36.28.578        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000009
 ABT =01  ADR =0001  ABN =000066  ACT =03  STATUS = 00001000   TLC = 0021
     001 055073065072O2D 0125016301450162005S  AUA%A+A]    USER-
     002 062072065061O6B 0142016201450141015S  A7A]A+A6A$  BREAK
     003 02D03102006606F 0055006100400146015S      [ 5A-A.  -1 FO
     004 06C06C06F077073 0154015401570167016S  A=A=A.A&A%  LLOWS
     005 01F000000000000 0037000000000000000S    4


10.36.28.578        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000010
 ABT =02  ADR =0001  ABN =000067  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 55111620252455201423    INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0


10.36.31.717        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000011
 ABT =00  ADR =0001  ABN =000000  ACT =03  STATUS = 01000000   TLC = 0000
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 1 of 11)

```
10.36.31.737        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000012
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010010B0 4060100000010001020O  FCACK


10.36.31.763        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000013
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001OC0 4060100000010001030O  FCACK


10.36.31.776        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000014
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002
     001 830001001001OC0 4060000100010001030O  FCBRK
     002 000000000000000 0000000000000000000


10.36.31.786        NETPUT  (030612)   HA =024534   TA =024535                          MSG NO. 000015
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8301000010000O0 4060040000010000000O  FCRST


10.36.31.787        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000016
 ABT =01  ADR =0001  ABN =000068  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4248504BB5CB6D 55022205011355345555   BREAK 1    4$  ; 6
     002 000000000000000 0000000000000000000              P


10.36.31.787        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000017
 ABT =02  ADR =0001  ABN =000069  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4939055485O313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000              0


10.36.33.221        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000018
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010011O0 4060100000010001040O  FCACK


10.36.33.226        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000019
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010011A0 4060100000010001050O  FCACK


10.36.42.633        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000020
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0020
     001 055073065072O2D 0125016301450162005S  AUA%A+A]    USER-
     002 062072065061O6B 0142016201450141015S  A7A]A+A6A$  BREAK
     003 02D032020066O6F 0055006200400146015S     ] 5A-A.  -2 FO
     004 06C06C06F077073 0154015401570167016S  A=A=A.A&A%  LLOWS


10.36.42.640        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000021
 ABT =01  ADR =0001  ABN =000070  ACT =03  STATUS = 00001000   TLC = 0021
     001 055073065072O2D 0125016301450162005S  AUA%A+A]    USER-
     002 062072065061O6B 0142016201450141015S  A7A]A+A6A$  BREAK
     003 02D032020066O6F 0055006200400146015S     ] 5A-A.  -2 FO
     004 06C06C06F077073 0154015401570167016S  A=A=A.A&A%  LLOWS
     005 01F000000000000 0037000000000000000S   4


10.36.42.640        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000022
 ABT =02  ADR =0001  ABN =000071  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4939055485O313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000              0


10.36.43.082        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000023
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010011B0 4060100000010001060O  FCACK
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 2 of 11)

```
10.36.43.086        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000024
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010011C0 4060100000010010700  FCACK


10.36.45.698        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000025
 ABT =00  ADR =0001  ABN =000000  ACT =03  STATUS = 01000000   TLC = 0000


10.36.45.705        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000026
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002
     001 8300020010011C0 4060000200010010700  FCBRK
     002 0000000000000000 0000000000000000000


10.36.45.710        NETPUT  (030612)   HA =024534   TA =024535                          MSG NO. 000027
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8301000010000000 4060040000010000000  FCRST


10.36.45.710        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000028
 ABT =01  ADR =0001  ABN =000072  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4248504BB5DB6D 55022205011355355555   BREAK 2    4$  ;]6
     002 0000000000000000 0000000000000000000              P


10.36.45.710        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000029
 ABT =02  ADR =0001  ABN =000073  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520 1423   INPUT PLS  4  UKP1
     002 0000000000000000 0000000000000000000              0


10.36.46.233        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000030
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010012O0 4060100000010011000  FCACK


10.36.46.237        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000031
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010012A0 4060100000010011100  FCACK              $


10.36.57.690        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000032
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000010   TLC = 0017
     001 04306106E063065 0103014101560143 0145  ACA6A,A8A+  CANCE
     002 06C020074068069 0154004001640150 0151  A= 5A"A/A(  L THI
     003 07302006C06906E 0163004001540151 0156  A% 5A=A(A,  S LIN
     004 065026000000000 0145004600000000000  A+ -         E&


10.36.57.719        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000033
 ABT =02  ADR =0001  ABN =000074  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520 1423   INPUT PLS  4  UKP1
     002 0000000000000000 0000000000000000000              0


10.36.58.189        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000034
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010012B0 4060100000010011200  FCACK              (


10.37.08.584        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000035
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0023
     001 0420720650610 6B 0102016201450141 0153  ABA]A+A6A$  BREAK
     002 02D06B065079020 0055015301450171 0040    A$A+A? 5  -KEY
     003 0620720650610 6B 0142016201450141 0153  A7A]A+A6A$  BREAK
     004 02006606F06C06C 0040014601570154 0154   5A-A.A=A=   FOLL
     005 06F077073000000 0157016701630000 0000  A.A&A%      OWS
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 3 of 11)

```
10.37.08.591        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000036
 ABT =01  ADR =0001  ABN =000075  ACT =03  STATUS = 00001000   TLC = 0024
     001 042072065061068 0102016201450141015 3  ABA]A+A6A$  BREAK
     002 02D068065079020 0055015301450171004 0    A$A+A? 5  -KEY
     003 062072065061068 0142016201450141015 3  A7A]A+A6A$  BREAK
     004 02006606F06C06C 0040014601570154015 4   5A-A.A=A=   FOLL
     005 06F07707301F000 0157016701630037000 0  A.A&A% 4     OWS


10.37.08.591        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000037
 ABT =02  ADR =0001  ABN =000076  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520142 3   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000 0              0


10.37.09.103        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000038
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001200 4060100000010001130 0  FCACK             ,


10.37.09.108        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000039
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001300 4060100000010001140 0  FCACK               0


10.37.14.286        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010      MSG NO. 000040
 ABT =00  ADR =0001  ABN =000000  ACT =03  STATUS = 01000000   TLC = 0000


10.37.14.297        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000041
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002
     001 830001001001300 4060000100010001140 0  FCBRK             0
     002 000000000000000 0000000000000000000


10.37.14.304        NETPUT  (030612)   HA =024534   TA =024535                          MSG NO. 000042
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830100001000000 4060040000010000000 0  FCRST


10.37.14.304        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000043
 ABT =01  ADR =0001  ABN =000077  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4248504BB5CB6D 5502220501135534555 5   BREAK 1    4$  ; 6
     002 000000000000000 0000000000000000000 0              P


10.37.14.304        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000044
 ABT =02  ADR =0001  ABN =000078  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520142 3   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000 0              0


10.37.16.419        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000045
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001340 4060100000010001150 0  FCACK               4


10.37.16.432        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000046
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001380 4060100000010001160 0  FCACK               8


10.37.26.483        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010      MSG NO. 000047
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000010   TLC = 0016
     001 042072065061068 0102016201450141015 3  ABA]A+A6A$  BREAK
     002 02D068065079020 0055015301450171004 0    A$A+A? 5  -KEY
     003 063061068E063065 0143014101560143014 5  A8A6A,A8A+  CANCE
     004 06C000000000000 0154000000000000000 0  A=          L
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 4 of 11)

```
10.37.26.493        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000048
 ABT =02  ADR =0001  ABN =000079  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520142 3   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000              0


10.37.27.425        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000049
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001 0013C0 4060100000010001 1700  FCACK             <


10.37.32.677        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000050
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 800062001000000 4000014200010000000  INTRUSR       B


10.37.32.681        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000051
 ABT =01  ADR =0001  ABN =000080  ACT =04  STATUS = 00000000   TLC = 0020
     001 B4265005340512D 5502312001232305045 5   BYPASSED   4&P 4E
     002 000000000000000 0000000000000000000              P


10.37.32.681        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000052
 ABT =02  ADR =0001  ABN =000081  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520142 3   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000000              0


10.37.33.259        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000053
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001400 4060100000010001 2000  FCACK             @


10.37.33.270        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063      MSG NO. 000054
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS\= 00000000   TLC = 0001
     001 830200001001440 4060100000010001 2100  FCACK             D


10.37.54.511        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010      MSG NO. 000055
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10000000   TLC = 0100


10.37.54.527        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063      MSG NO. 000056
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0100
     001 054079070065061 0124017101600145014 1  ATA?A#A+A6  TYPEA
     002 068065061064020 0150014501410144004 0  A/A+A6A9 5  HEAD
     003 074065073074020 0164014501630164004 0  A"A+A%A" 5  TEST
     004 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     005 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     006 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     007 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     008 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     009 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     010 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     011 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     012 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     013 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     014 061061061061061 0141014101410141014 1  A6A6A6A6A6  AAAAA
     015 061061061020062 0141014101410040014 2  A6A6A6 5A7  AAA B
     016 062062062062062 0142014201420142014 2  A7A7A7A7A7  BBBBB
     017 062062062062062 0142014201420142014 2  A7A7A7A7A7  BBBBB
     018 062062062062062 0142014201420142014 2  A7A7A7A7A7  BBBBB
     019 062062062062062 0142014201420142014 2  A7A7A7A7A7  BBBBB
     020 062062062062062 0142014201420142014 2  A7A7A7A7A7  BBBBB


10.37.54.534        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000057
 ABT =01  ADR =0001  ABN =000082  ACT =03  STATUS = 00001000   TLC = 0101
     001 054079070065061 0124017101600145014 1  ATA?A#A+A6  TYPEA
     002 068065061064020 0150014501410144004 0  A/A+A6A9 5  HEAD
     003 074065073074020 0164014501630164004 0  A"A+A%A" 5  TEST
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 5 of 11)

```
     004 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     005 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     006 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     007 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     008 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     009 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     010 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     011 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     012 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     013 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     014 061061061061061 0141014101410141 0141   A6A6A6A6A6  AAAAA
     015 061061061020062 0141014101400142 0141   A6A6A6 5A7  AAA B
     016 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     017 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     018 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     019 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     020 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     021 01F000000000000 0037000000000000 0000   4


10.38.00.212        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000058
 ABT =00  ADR =0001  ABN =000000  ACT =00  STATUS = 10000000   TLC = 0100


10.38.00.271        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063       MSG NO. 000059
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0100
     001 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     002 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     003 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     004 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     005 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     006 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     007 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     008 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     009 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     010 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     011 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     012 062062020063063 0142014200400143 0143   A7A7 5A8A8  BB CC
     013 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     014 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     015 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     016 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     017 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     018 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     019 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     020 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC


10.38.00.319        NETPUT  (030612)   HA =000314   TA =000365                        MSG NO. 000060
 ABT =01  ADR =0001  ABN =000083  ACT =03  STATUS = 00001000   TLC = 0101
     001 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     002 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     003 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     004 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     005 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     006 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     007 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     008 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     009 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     010 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     011 062062062062062 0142014201420142 0142   A7A7A7A7A7  BBBBB
     012 062062020063063 0142014200400143 0143   A7A7 5A8A8  BB CC
     013 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     014 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     015 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     016 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     017 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     018 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     019 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     020 063063063063063 0143014301430143 0143   A8A8A8A8A8  CCCCC
     021 01F000000000000 0037000000000000 0000   4
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 6 of 11)

```
10.38.05.028        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000061
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10000000   TLC = 0100


10.38.05.070        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063       MSG NO. 000062
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0100
     001 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     002 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     003 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     004 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     005 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     006 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     007 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     008 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     009 063020064064064 0143004001440144 A8 5A9A9A9  C DDD
     010 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     011 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     012 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     013 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     014 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     015 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     016 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     017 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     018 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     019 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     020 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD


10.38.05.086        NETPUT  (030612)   HA =000314   TA =000365                          MSG NO. 000063
 ABT =01  ADR =0001  ABN =000084  ACT =03  STATUS = 00001000   TLC = 0101
     001 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     002 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     003 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     004 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     005 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     006 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     007 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     008 063063063063063 0143014301430143 A8A8A8A8A8  CCCCC
     009 063020064064064 0143004001440144 A8 5A9A9A9  C DDD
     010 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     011 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     012 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     013 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     014 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     015 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     016 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     017 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     018 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     019 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     020 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     021 01F000000000000 0037000000000000000  4


10.38.11.191        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000064
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10000000   TLC = 0100


10.38.11.200        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063       MSG NO. 000065
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0100
     001 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     002 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     003 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     004 064064064064064 0144014401440144 A9A9A9A9A9  DDDDD
     005 064064064020065 0144014401440040145 A9A9A9 5A+  DDD E
     006 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
     007 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
     008 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
     009 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
     010 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
     011 065065065065065 0145014501450145 A+A+A+A+A+  EEEEE
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 7 of 11)

```
     012 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     013 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     014 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     015 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     016 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     017 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     018 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     019 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     020 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE


10.38.31.619        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000066
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001480 4060100000010012200 0148  FCACK             H


10.38.31.623        NETPUT  (030612)   HA =000510   TA =000750                            MSG NO. 000067
 ABT =01  ADR =0001  ABN =000085  ACT =03  STATUS = 00001000   TLC = 0101
     001 064064064064064 0144014401440144 0144  A9A9A9A9A9  DDDDD
     002 064064064064064 0144014401440144 0144  A9A9A9A9A9  DDDDD
     003 064064064064064 0144014401440144 0144  A9A9A9A9A9  DDDDD
     004 064064064064064 0144014401440144 0144  A9A9A9A9A9  DDDDD
     005 064064064020065 0144014401400145 0144  A9A9A9 5A+  DDD E
     006 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     007 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     008 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     009 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     010 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     011 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     012 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     013 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     014 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     015 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     016 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     017 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     018 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     019 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     020 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     021 01F000000000000 0037000000000000000   4


10.38.31.624        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000068
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10000000   TLC = 0100


10.38.31.634        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063       MSG NO. 000069
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0100
     001 065065065065065 0145014501450145 0145  A+A+A+A+A+  EEEEE
     002 065065065020066 0145014501400146 0145  A+A+A+ 5A-  EEE F
     003 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     004 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     005 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     006 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     007 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     008 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     009 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     010 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     011 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     012 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     013 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     014 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     015 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     016 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     017 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     018 066066066066066 0146014601460146 0146  A-A-A-A-A-  FFFFF
     019 066066066066020 0146014601460040 0146  A-A-A-A- 5  FFFF
     020 067067067067067 0147014701470147 0147  A*A*A*A*A*  GGGGG
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 8 of 11)

```
10.38.37.832        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000070
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010014C0 4060100000010001 2300  FCACK             L


10.38.37.836        NETPUT  (030612)   HA =000510   TA =000750                            MSG NO. 000071
 ABT =01  ADR =0001  ABN =000086  ACT =03  STATUS = 00001000   TLC = 0101
     001 065065065065065 0145014501450145 0145  A+A+A+A+A+A+  EEEEE
     002 065065065020066 0145014501450040 0146  A+A+A+ 5A-    EEE F
     003 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     004 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     005 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     006 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     007 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     008 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     009 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     010 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     011 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     012 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     013 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     014 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     015 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     016 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     017 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     018 066066066066066 0146014601460146 0146  A-A-A-A-A-    FFFFF
     019 066066066066020 0146014601460140 0146  A-A-A-A- 5    FFFF
     020 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     021 01F000000000000 0037000000000000 0000   4


10.38.37.874        NETGET  (030270)  ACN =0001  HA =000314  TA =000365  TLMAX =0063       MSG NO. 000073
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0080
     001 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     002 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     003 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     004 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     005 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     006 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     007 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     008 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     009 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     010 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     011 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     012 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     013 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     014 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     015 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     016 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG


10.38.44.616        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000074
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010015O0 4060100000010001 2400  FCACK             P


10.38.44.620        NETPUT  (030612)   HA =000510   TA =000750                            MSG NO. 000075
 ABT =01  ADR =0001  ABN =000087  ACT =03  STATUS = 00001000   TLC = 0081
     001 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     002 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     003 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     004 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     005 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     006 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     007 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     008 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     009 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     010 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
     011 067067067067067 0147014701470147 0147  A*A*A*A*A*    GGGGG
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 9 of 11)

```
     012 067067067067067 014701470147014701 47  A*A*A*A*A*A*  GGGGG
     013 067067067067067 014701470147014701 47  A*A*A*A*A*A*  GGGGG
     014 067067067067067 014701470147014701 47  A*A*A*A*A*A*  GGGGG
     015 067067067067067 014701470147014701 47  A*A*A*A*A*A*  GGGGG
     016 067067067067067 014701470147014701 47  A*A*A*A*A*A*  GGGGG
     017 01F000000000000 00370000000000000000   4


10.38.44.621        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000076
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001540 4060100000010001 2500  FCACK             T


10.38.44.661        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000077
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001580 4060100000010001 2600  FCACK             X


10.38.45.646        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000078
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010015C0 4060100000010001 2700  FCACK


10.38.57.587        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000079
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0000


10.38.57.594        NETPUT  (030612)   HA =000314   TA =000365                           MSG NO. 000080
 ABT =01  ADR =0001  ABN =000088  ACT =03  STATUS = 00001000   TLC = 0001
     001 01F000000000000 00370000000000000000   4


10.38.57.595        NETPUT  (030612)   HA =000314   TA =000365                           MSG NO. 000081
 ABT =02  ADR =0001  ABN =000089  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520 1423   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000 0000              0


10.38.58.150        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000082
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001600 4060100000010001 3000  FCACK             @


10.38.58.178        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000083
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001640 4060100000010001 3100  FCACK             D


10.39.13.806        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000084
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0009
     001 04907402007706F 0111016400400167 0157  AIA" 5A&A.  IT W0
     002 07206B073021000 0162015301630041 0000  A]A$A% 6    RKS!


10.39.13.813        NETPUT  (030612)   HA =000314   TA =000365                           MSG NO. 000085
 ABT =01  ADR =0001  ABN =000090  ACT =03  STATUS = 00001000   TLC = 0010
     001 04907402007706F 0111016400400167 0157  AIA" 5A&A.  IT W0
     002 07206B07302101F 0162015301630041 0037  A]A$A% 6 4  RKS!


10.39.13.813        NETPUT  (030612)   HA =000314   TA =000365                           MSG NO. 000086
 ABT =02  ADR =0001  ABN =000091  ACT =04  STATUS = 00000000   TLC = 0020
     001 B49390554B50313 5511162025245520 1423   INPUT PLS  4  UKP1
     002 000000000000000 0000000000000000 0000              0


10.39.14.349        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000087
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001001680 4060100000010001 3200  FCACK             H
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 10 of 11)

```
10.39.14.353        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000088
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010016C0 4060100000010001330O  FCACK             L


10.39.18.475        NETGETL (030304)  ALN =0001  HA =000314  TA =000365  TLMAX =0010       MSG NO. 000089
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00010000   TLC = 0005
     001 04504E04404304E 0105011601040103011 6  AEANADACAN  ENDCN


10.39.18.482        NETPUT  (030612)   HA =024534   TA =024535                            MSG NO. 000090
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002
     001 630600001000000 3060300000010000000O  CONEND      C
     002 2411ADB6DB40000 1101065555555500000O  IAF          A [M4


10.39.18.653        NETGETL (030304)  ALN =0000  HA =024534  TA =024535  TLMAX =0063       MSG NO. 000091
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 634600001000000 3064300000010000000O  CONENDN     CF


10.39.24.187        NETOFF  (025312)       DATE =82/11/16                                MSG NO. 000092
```

Figure I-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 11 of 11)

```
NAM STATISTICS GATHERING STARTED
NETON   DATE  82/11/16.   TIME 10.35.16.

NAM STATISTICS GATHERING TERMINATED
NETOFF  DATE  82/11/16.   TIME 10.39.24.

CPU TIME USED:          0.316 SEC


NUMBER OF PROCEDURE CALLS
     NETGET                  6
     NETGETL                51
     NETPUT                 33
     NETWAIT                81

NUMBER OF WORKLIST TRANSFER ATTEMPTS
     SUCCESSFUL             79

NUMBER OF INPUT/OUTPUT BLOCKS TRANSFERRED
     INPUT   ABT=0           9
     INPUT   ABT=1           5
     INPUT   ABT=2           9
     INPUT   ABT=3          34
     OUTPUT  ABT=1          15
     OUTPUT  ABT=2          12
     OUTPUT  ABT=3           6

NUMBER OF ERRORS
```

Figure I-5. Statistical File Listing for Sample FORTRAN Program

INDEX

COMMENT SHEET

MANUAL TITLE: Network Products Network Access Method Version 1/
Communications Control Program Version 3 Reference Manual

PUBLICATION NO.: 60499500 REVISION: P

NAME:

COMPANY:

STREET ADDRESS:

CITY: STATE: ZIP CODE:

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

___ Please reply ___ No reply necessary

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division

215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

FOLD

FOLD