60499500

CONTROL DATA

NETWORK PRODUCTS

NETWORK ACCESS METHOD VERSION 1 HOST APPLICATION PROGRAMMING 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 This manual, which was previously known as the NAM Reference Manual, is no longer applicable to products operating under NOS 1.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.
R (09/30/83)	Revised at PSR level 596 to reflect release of NAM Version 1.6 and CCP Version 3.6, supporting multiple-host networks. This is a complete reprint.
S (09/19/84)	Revised at PSR level 617 to reflect release of NAM Version 1.7 and CCP Version 3.7 to document support of a 3270 bisynchronous terminal class and miscellaneous technical corrections.
T (09/30/85)	Revised at PSR level 642 to reflect release of NAM Version 1.8 and CCP Version 3.8. This manual was previously known as the NAM Version 1/CCP Version 3 Host Application Programming Reference Manual. Miscellaneous technical changes are included.
U (12/16/85)	Revised at PSR level 647 to reflect release of NAM Version 1.8, CCP Version 3.8, and CDCNET Version 1.0. Miscellaneous technical corrections are included.
V (07/31/86)	Revised at PSR level 664 to reflect release of NAM Version 1.8, CCP Version 3.8, and CDCNET Version 1.1. Miscellaneous technical corrections are included.
W (04/23/87)	Revised at PSR level 678 to reflect release of NAM Version 1.8, CCP Version 3.8, and CDCNET Version 1.2. Miscellaenous technical corrections are included.

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

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Technology and Publications Division
P. O. BOX 3492
SUNNYVALE, CALIFORNIA 94088-3492

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	W
iii/iv	W
v	V
vi	U
vii/viii	V
ix thru xii	V
xiii/xiv	V
xv	T
1-1	R
1-2 thru 1-6	U
1-7	V
1-8	V
1-8.1	U
1-8.2	U
1-9 thru 1-14	T
2-1	U
2-2	R
2-3	R
2-4	T
2-5	R
2-6	T
2-7	R
2-8	T
2-9	R
2-10	U
2-11	T
2-12 thru 2-14	S
2-15 thru 2-18	U
2-19	V
2-20	U
2-21	S
2-22 thru 2-25	T
2-26	W
2-27	T
2-28	W
2-29	T
2-30 thru 2-33	W
2-34	V
2-35	U
2-36 thru 2-39	T
3-1	U
3-2 thru 3-6	V
3-7 thru 3-10	U
3-11	V
3-12	V
3-12.1/3-12.2	U
3-13 thru 3-16	T
3-17	V
3-18 thru 3-21	T
3-22	V
3-23	W
3-24	W
3-25	T
3-26	V
3-27	W
3-28	W
3-29	V
3-30	V
3-30.1	V
3-30.2	V
3-31	V
3-32 thru 3-44	T
3-45	V
3-46	V
3-47 thru 3-50	T
3-51 thru 3-53	V
3-54 thru 3-57	T
3-58	V
3-59	V
3-60	U
3-61	W
3-62 thru 3-68	V
3-68.1 thru 3-68.6	V
3-69 thru 3-79	U
3-80	V
3-81	U
4-1	R
4-2	R
4-3	W
4-4	W
4-4.1/4-4.2	W
4-5 thru 4-10	V
4-10.1	V
4-10.2	V
4-11 thru 4-15	T
4-16	V
4-17 thru 4-19	T
5-1	R
5-2	V
5-3	W
5-4	R
5-5	R
5-6	T
5-7 thru 5-11	R
5-12	T
5-13	R
5-14	T
5-15	T
5-16	R
5-17	T
6-1	T
6-2	V
6-3	T
6-4	T
6-5	S
6-6	S
6-7	R
6-8	R
6-9	T
6-10	W
6-11	R
6-12	V
6-13 thru 6-15	R
6-16	V
6-17	R
7-1 thru 7-15	R
7-16	T
7-17 thru 7-24	R
7-25	T
7-26 thru 7-38	R
8-1	R
8-2 thru 8-12	V
8-13	W
8-14 thru 8-34	V
8-34.1	V
8-34.2	V
8-34.3/8-34.4	V
8-35 thru 8-66	T
A-1 thru A-3	R
A-4	S
A-5 thru A-19	R
A-20 thru A-23	S
A-24 thru A-32	T
A-33 thru A-36	R
A-37	T
A-38	T
A-39	S
A-40 thru A-46	R
A-47	S
A-48	R
B-1	W
B-2	W
B-2.1/B-2.2	W
B-3	T
B-4	T
B-5	V
B-6	V
B-7 thru B-9	T
C-1 thru C-13	U
D-1	V
D-2	V
Index-1 thru -6	V
Comment Sheet/Mailer	W
Back Cover	-

PREFACE

This manual supplies reference information to both Network Access Method (NAM) Version 1.7 and Communications Control Program (CCP) Version 3.7 users, typically either 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 manual describes how application programs interface to the computer network. The NAM 1/CCP 3 Terminal Interface reference manual describes how the terminal user gains access to these applications. Also, this manual 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 180 Computer Systems; 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 a 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

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 contains a FORTRAN program using AIP; section 8 describes QTRM. Section 9 describes network failure and techniques of recovery.

Additional reference information for the Communications Control Program can be found in other network product and operating system publications. Use table 0-1 to locate this information.

TABLE 0-1. LOCATION OF CCP REFERENCE INFORMATION

Information	Manual That Contains Information					
	NOS Version 2 Administration Handbook	NAM 1/CCP 3 Terminal Interfaces Reference Manual	NOS Version 2 System Analysis Handbook	Communications Control Program Version 3 Diagnostic Handbook	NOS Version 2 Operations Handbook	Communications Control Program Internal Maintenance Specification†
CCP overview, concepts, and functions		X				
Character sets		X				
CCP glossary						X
Mnemonics						X
Statistics	X					
Halt Codes				X		

TABLE 0-1. LOCATION OF CCP REFERENCE INFORMATION (Contd)

Information	Manual That Contains Information					
	NOS Version 2 Adminis-tration Handbook	NAM 1/CCP 3 Terminal Interfaces Reference Manual	NOS Version 2 System Analysis Handbook	Communications Control Pro-gram Version 3 Diagnostic Handbook	NOS Version 2 Opera-tions Handbook	Communications Control Program Internal Maintenance Specification†
Diagnostics				X		
Customer Engineering error messages				X		
Dump information				X		
NPU operating instructions			X		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

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

The following manuals are of primary interest:

Publication	Publication Number
Network Products Network Access Method Version 1 Network Definition Language Reference Manual	60480000
Network Products Network Access Method Version 1/ Communications Control Program Version 3 Terminal Interfaces Reference Manual	60480600
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
Communications Control Program Version 3 Diagnostic Handbook	60471500
COMPASS Version 3 Reference Manual	60492600
COBOL Version 5 Reference Manual	60497100
CYBER Cross System Version 1 Build Utilities Reference Manual	60471200
CYBER Cross System Version 1 Macro Assembler Reference Manual	96836500
CYBER Cross System Version 1 Micro Assembler Reference Manual	96836400
CYBER Cross System Version 1 PASCAL Reference Manual	96836100
FORTRAN Version 5 Reference Manual	60481300
Hardware Performance Analyzer (HPA) User Reference Manual	60459460
Message Control System Version 1 Reference Manual	60480300
NOS Version 2 Diagnostic Index	60459390
NOS Version 2 Installation Handbook	60459320
NOS Version 2 Manual Abstracts	60485500
NOS Version 2 Administration Handbook	60459840
NOS Version 2 Operations Handbook	60459310
NOS Version 2 Analysis Handbook	60459300
Network Products Remote Batch Facility Version 1 Reference Manual	60499600
Software Publications Release History	60481000
TAF Version 1 Reference Manual	60459500
2551-1, 2551-2, 2552-2 Network Processor Unit Hardware Reference Manual	60472800
2560 Series Synchronous Communications Line Adapter Hardware Maintenance Manual	74700700

Publication	Publication Number
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).
$\boxed{\text{cr}}$	The boxed cr symbol represents the terminal key that causes message transmission; usually, this key causes a carriage return operation. Transmission keys are described in more detail in section 2.

Unless otherwise specified, all references to numbers are to decimal values, 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 CDC network consists of a computer network, a communications network, and a services network.

COMPUTER NETWORK

The computer network includes host computer systems packet-switching networks (PSNs), terminals, and the host software associated with network communications.

Each component of the computer network provides input, output, control, or storage resources to the services and communications network. The primary host communication software is called the Network Access Method (NAM).

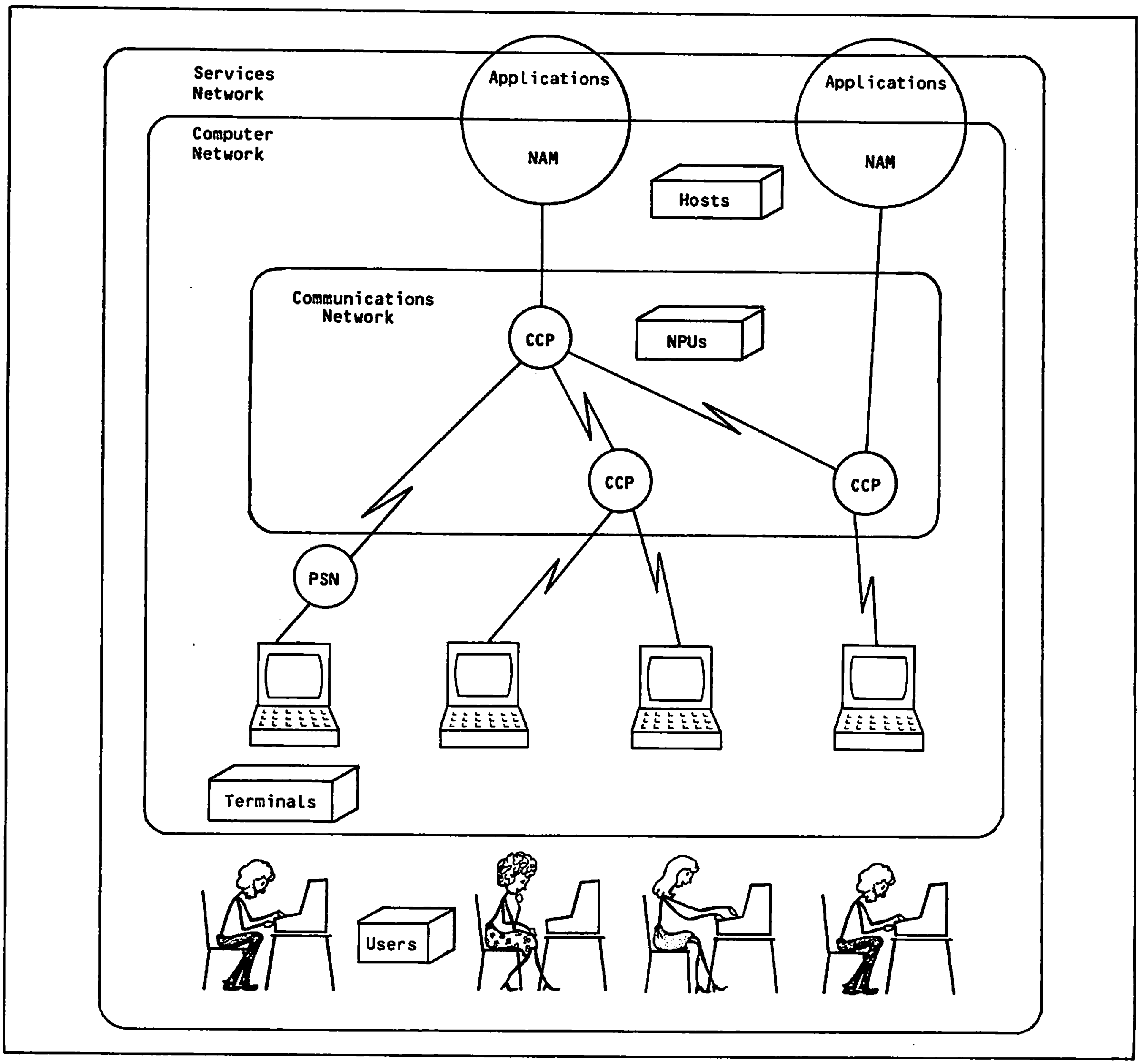


Figure 1-1. Overview of a CDC Network

COMMUNICATIONS NETWORK

The communications network includes network processing units (NPUs) and the connecting communication lines needed to transport blocks of data between host computers and terminals. The primary CDC software in an NPU is called the Communications Control Program (CCP).

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.

SERVICES NETWORK

The services network consists of the network application programs in each host computer and the users of those programs. Each application program gives the terminal user or another application a specific data processing capability.

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 software elements, 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 with the terminals while the Network Access Method (NAM) communicates with application programs. Refer to figure 1-2 while reading the remainder of this overview section on network products.

The network host 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).

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

Network host software, 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

CYBER Cross System

NETWORK ACCESS METHOD

The Network Access Method is the primary network host software. NAM interfaces between applications in the same host or between applications and the Communications Control Program in an NPU.

Because the connections among NPUs can become extremely complex, the Network Access Method acts as an interface between host computer software at one end of the network and the terminals at the other end.

A simple front-end NPU configuration appears the same through the Network Access Method as a more complex linkage system; message routing by the host computer is performed in the same manner for both configurations. The physical and logical configuration of the elements involved in Network Access Method operation is described in the Network Definition Language reference manual (listed in the preface).

The host computer executes CDC-written or site-written service programs called application programs that are connected to the network via the Network Access Method (NAM). An application program can communicate with other application programs or service terminals connected to the network. All connections to the network are established by a portion of the network software called the Network Validation Facility, and the flow of data and processing along them is controlled through NAM.

NAM incorporates the following features:

- It is equally suitable for application programs written in COMPASS or high-level languages, such as FORTRAN.
- It imposes no data structures on an application program.
- It provides a way to handle unpredictable events, such as terminal operator interrupts.
- It provides complete isolation of network communications from the operating system.
- It supports distinct classes of terminals by normalizing data formats and optionally performing code conversion. Seventeen classes are defined by CDC; additional classes can be defined by sites that provide their own supporting software.
- It permits an application program to support clusters of real terminal devices as if the devices were separately addressable logical entities called virtual terminals. Virtual terminals are described at the end of this section.

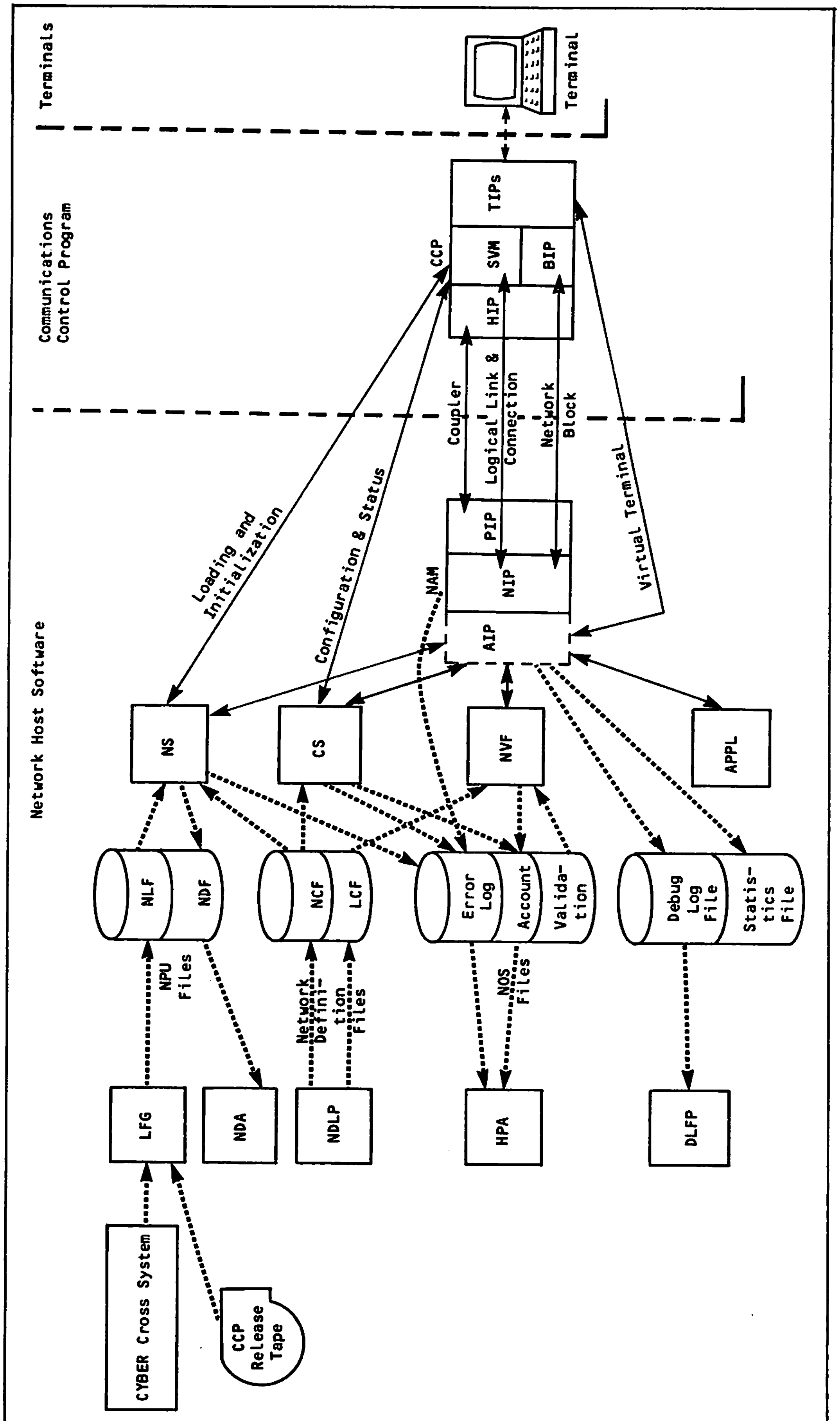


Figure 1-2. The Interfaces Between the Network Product Elements

Basic services provided by NAM include:

- NAM establishes message paths (logical connections) between an application program and terminals or between two applications (provided both parties have the correct network access security permissions).

- NAM breaks logical connections when asked to by the application program or the terminal, or when network conditions make it necessary (for example, when a network shutdown occurs).

- After logical connections have been established, NAM passes incoming messages to the application, and accepts and forwards outgoing messages from the application.

- NAM queues incoming messages until the application program requests them. This allows the application to service its connections with terminals and other applications in any desired order.

- NAM provides the application program with its own set of protocols, making knowledge of detailed network protocols unnecessary.

- For incoming traffic, NAM allows the application program to group terminals with similar or related processing needs.

- NAM queues outgoing messages to regulate data flow through the network.

- NAM detects inactivity on any interactive data path and reports the condition to the application program.

- NAM resolves resource contention among application programs.

An installation option is available to log message traffic for application program debugging. A second installation option permits the logging of application program and message traffic statistics.

NAM consists of four major modules:

Peripheral Interface Program

Network Interface Program

Application Interface Program

Queued Terminal Record Manager

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 or for file transfer. PIP also can detect when a local NPU needs initializing. If the NPU cannot start its own loading, 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.

Each application program can 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 can 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 and buffers that resides in the application program's field length and provides an interface to NIP and the network. This manual is primarily concerned with the use of AIP.

AIP statements are provided so that the application program can connect to and disconnect from the network. AIP statements also control information exchange between the application program and NAM buffers. This information can be data, or it can be supervisory messages that coordinate the application's execution with events that have occurred in the network. NAM might 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

The Queued Terminal Record Manager (QTRM) is a set of subprograms that resides in the application program's field length and provides a high level procedural interface to the network. This package permits indirect use of a subset of AIP's features by programs with unsophisticated communications requirements. This utility permits programs to have a communications interface functionally similar to their mass storage interface. QTRM is discussed in section 8 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), a batch utility, translates this configuration and prepares a network configuration file (NCF) and a local configuration file (LCF).

The NCF contains configuration information required by the network.

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

The NDL is described in the Network Definition Language reference manual listed in the preface.

NETWORK SUPERVISOR

The Network Supervisor (NS) executes as a NAM application. It interfaces between the NPUs and CCP program files in the host. NS loads an NPU on request with the appropriate copy of the Communications Control Program from the host's network load file (NLF). NS also saves NPU dumps in the host's network dump file (NDF). The load and dump files are shown in figure 1-2.

The host operator can obtain status information for NPU loading or dumping operations involving the copy of NS in the operator's host. More than one host can run a copy of NS; so that NS can load NPUs which are not accessible from a specific host.

COMMUNICATION SUPERVISOR

The Communication Supervisor (CS) program executes as a NAM application. It can communicate with the network operators (NOP). CS allows a network operator at a terminal (an NPU operator or a diagnostic operator [DOP]) or at a host console (a host operator [HOP]) to obtain and change the status of network elements under its supervision, to communicate with users at terminals, and to run diagnostics. CS also responds to requests for network configuration data from an NPU.

CS can run in one or more hosts. It also assists the NPUs by providing them with terminal configuration information from the network configuration file.

NETWORK VALIDATION FACILITY

The Network Validation Facility (NVF) also executes as a NAM application. It validates the terminal user's access to the host and an application program's access to the computer network. 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 to NVF.

NVF verifies such terminal user information as family name, user name, and password. Before a terminal user can access an application program, successful login must occur. When login is successfully completed, the Network Validation Facility causes NAM to notify the application program identified in the login sequence that a terminal requests connection.

The Network Validation Facility also performs switching between application programs. NVF causes terminal disconnection processing when disconnection is appropriate.

The Network Validation Facility controls application program and terminal access to the network, as follows:

- An application program wishing to communicate with terminals requests access to the network. This request is passed by NAM to the NVF for validation. (NVF also performs similar validation of terminal requests for host access.) Once NVF has determined that an application program or terminal is allowed to use the host's resources, it makes calls to NAM that create the logical connection for the transfer of data between the application program and the network. NVF also requests NAM to modify or delete these connections when terminal users request to communicate with other application programs or leave the network.

- When an application program no longer desires to use the network, it calls another NAM procedure. This request also is passed to NVF, which causes NAM to delete all connections used for the application program - just as it does for a terminal or terminal device leaving the network.

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

The network dump analyzer (NDA) 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.

You can find more information about the NPU dump analyzer in the NOS Version 2 Analysis Handbook listed in the preface.

Load File Generator

The load file generator (LFG) 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.

You can find more information about the load file generator in the NOS Installation Handbook listed in the preface.

Debug Log File Processor

The debug log file processor (DLFP) converts the debug log file generated by the Application Interface Program into a printable report. The programmer can selectively list logged information through DLFP directives.

You can find more information about the debug log file processor in section 6 of this manual.

Hardware Performance Analyzer

A fourth utility program, the hardware performance analyzer (HPA), is part of the NOS operating system. This 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.

You can find more information about the hardware performance analyzer in the HPA reference manual listed in the preface.

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 be used by 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 Interfaces reference manual.

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.

The queue file transfer facility (QTF), which allows you to transfer queue files between hosts. The use of this feature is described in the NOS Version 1 Reference Set, Volume 3.

Permanent File Transfer Facility (PTF), which allows you to transfer permanent files between waits. The use of this feature is documented in the NOS Version 2 Reference Set, Volume 3.

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 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 memory 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 memory 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.

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 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
- System autostart module program (SAM-P)
- Service module (SVM)
- Host Interface Program (HIP)
- Terminal Interface Programs (TIPs)
- Link Interface Program (LIP)
- Block Interface Program (BIP)
- In-line and on-line diagnostics
- NPU 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 255x multiplexing 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

System Autostart Module

The system autostart module is an optional set of hardware and software that begins the loading of other CCP software from a host.

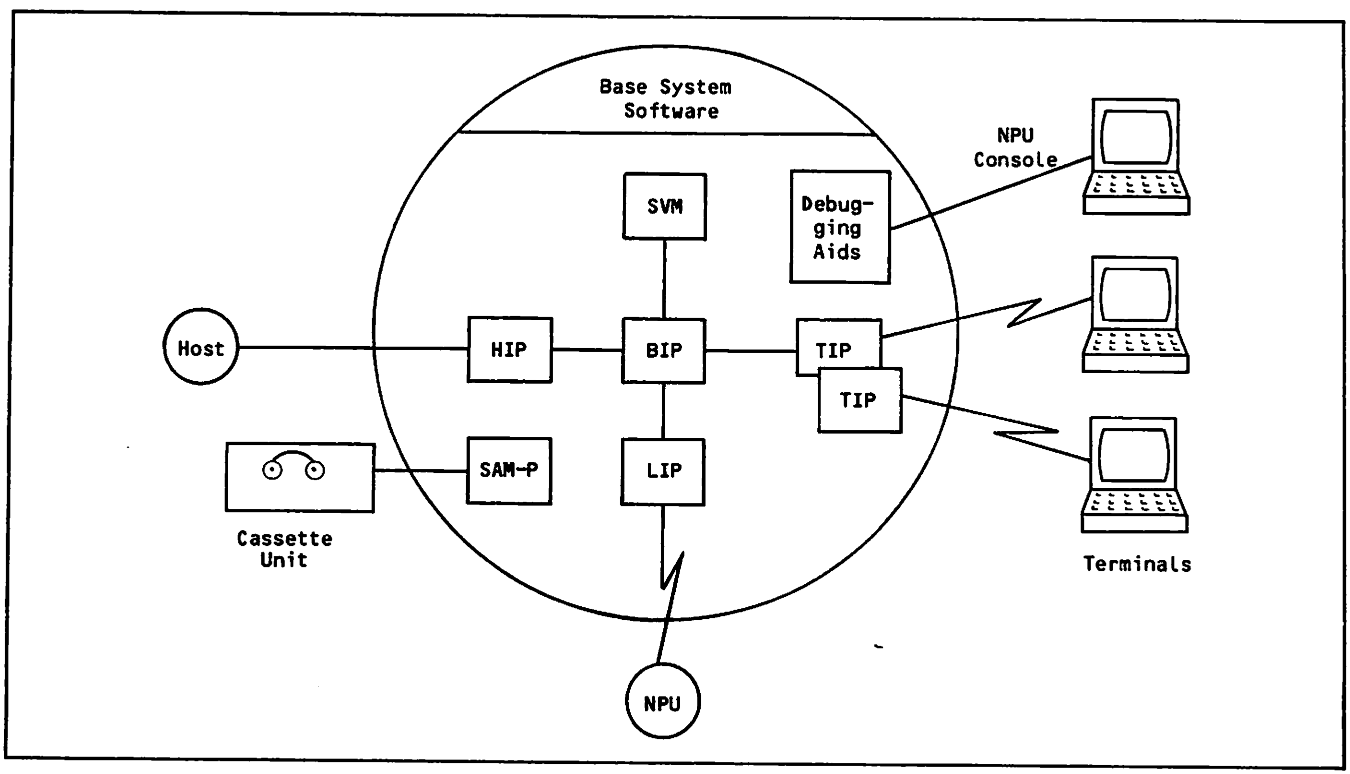


Figure 1-3. The Relationship Between the Parts of the Communications Control Program

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 commands from the host, called 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.

The TIP transforms application program data between its virtual terminal format and the format required by the transmission protocol and physical characteristics of the real terminals. CDC provides TIPs for these transmission protocols:

- Asynchronous communication lines
- Synchronous communication lines for mode 4 terminals
- Bisynchronous communication lines for terminals emulating the IBM HASP protocol
- X.25 packet and link level interfaces to a packet-switching network (PSN) via high-level data link control (HDLC) synchronous lines
- Bisynchronous communications lines for terminals emulating the IBM 2780/3780 protocol
- 3270 Bisynchronous communications (BSC) operating as multipoint data links

Eighteen classes of real terminals using these protocols are supported. Each terminal class has certain physical characteristics associated with it. These associated characteristics are determined by a terminal chosen as the archetype for the class, but can be changed by either the application program or the terminal operator. The terminal class initially used for a given real terminal is determined by the way the terminal is configured in the network configuration file; the network configuration file can also be used to change the characteristics initially associated with the terminal from those of the archetype terminal. The association of characteristics with a terminal is referred to in networks documentation as terminal definition or TERMDEF.

The terminal classes and archetype terminals for each class are listed at the end of this section.

This list includes only elements supported by released versions of standard CDC network software.

Sites can add site-written Terminal Interface Programs to extend CDC support to additional transmission protocols and terminal classes. This manual is concerned only with the transmission protocols and terminal classes supported by CDC. Information in this manual is valid for sites using extensions to CCP only to the extent that those modifications emulate the CDC-supported release version of CCP.

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.

NPU Console Debugging Aids

Debug aids provide test utilities for debugging programs, taking memory snapshots, and dumping the NPU during CCP 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.

THE PACKET SWITCHING NETWORK (PSN)

The packet switching network (PSN) is a value added network you may subscribe to either from a CDC or a foreign vendor who supports the X.25 CCITT recommendation (1980). Such networks are alternately referred to as public data networks (PDNs).

NAM CONCEPTS

NAM is used by both application programs and portions of the network software. The features of NAM permit programs to be written for the following types of communication applications:

- Time-sharing communication services. A single program provides this service when it interacts with each terminal during a given time period. The CDC-written Interactive Facility is an example of this type of application program.

- Transaction communication services. A single program provides this service when it creates a multi-threading interface for many terminals using many task routines. Each terminal can interact with many tasks or programs through queues maintained by the program providing the transaction service. The CDC-written Transaction Facility is an example of this type of application program.

- Teleprocessing communication services. A single program provides this service when it interacts with many terminals to perform a single teleprocessing task for each. No task queues are required. The CDC-written Terminal Verification Facility is an example of this type of application program.

VIRTUAL TERMINALS

The virtual terminal concept simplifies the procedure an application program must perform to service a terminal.

Device types are used in a request for connection from a terminal to an application (see section 3 for a discussion of connection processing). Device types currently defined are listed in table 1-1.

TABLE 1-1. DEVICE TYPES

Device Type	Terminal Device Defined
0	Console (interactive device)
1†	Card reader (passive device)
2†	Line printer, impact printer or nonimpact printer (passive device)
3†	Card punch (passive device)
4†	Plotter (passive device)
5	Another application program in the same host
6	Another application program in a different host
7 thru 11	Reserved for CDC use
12	Site-defined device

†Reserved for RBF use.

Every terminal device is either an interactive device (capable of both input and output) or a batch device (capable of either input or output). Because this is true of all physical terminals, certain functions of each terminal device type can be abstracted and treated in a similar manner for all terminals with devices of that type. These common functions constitute a virtual terminal. All references to terminals in this manual are to virtual terminals, unless otherwise specified.

The interactive virtual terminal concept makes it unnecessary for an application programmer to provide separate procedures to support differing implementations of one function on a variety of real terminals.

Any console or site-defined device (any device with a device type of 0 or 12) can be serviced as an interactive virtual terminal. An interactive virtual terminal has an input and output device which sends and receives logical lines of ASCII characters. These logical lines are transformed into or from physical lines of characters of the code set appropriate for the real terminal. This transformation is performed for the application program by the Communications Control Program of the network processing unit servicing the real terminal.

Real terminals can perform a wide variety of functions, but not all terminals can perform the same functions. The functions performed by an interactive virtual terminal are restricted to the subset of terminal functions that is common to all real interactive terminals. This restriction ensures efficient virtual terminal operation when the corresponding real terminal has the fewest capabilities.

When the application program must support functions for a real terminal that are not available through the interactive virtual terminal interface, the application program can:

- Embed control characters in the output text or scan for control characters in the input text. The application program must allow for control characters significant to or transformed by the network software in this instance.

- Transfer data to and from the terminal in transparent mode. In transparent mode, all transformations are inhibited and the application program has direct access to and responsibility for support of all real terminal functions. Transparent mode can be selected separately for input and output to the same virtual terminal.

Control characters and transparent mode are discussed in detail in section 2.

Logical lines that exceed the physical line length of the real terminal are folded into two or more physical lines on output to the terminal. The spacing of output lines can also be controlled with optional format effectors, described in section 2. Optional paging of output is possible, to avoid overwriting previous output until the previous output is acknowledged by the terminal operator.

LOGICAL CONNECTIONS

Just as the virtual terminal concept simplifies terminal servicing, the logical connection concept simplifies terminal addressing. In the network, when data passes between a virtual terminal and an application program, a message path or logical connection exists between the two. Conceptually, this is equivalent to the connection between two telephones used in a conversation. After a real terminal has gained network access, NAM logically connects each virtual terminal portion of it to one, and only one, application program at a time, although the virtual terminal can be switched from application to application as needed.

An application program, however, can be connected simultaneously to many virtual terminals. It is connected to each one by a separate and distinct logical connection. The application program identifies a particular terminal by specifying the logical connection between itself and the terminal. This is possible because a one-to-one association exists between the connection and the terminal. From the application programmer's point of view, it is convenient to talk of connection x (literally, message path x) when it would be more precise to say the virtual terminal at the other end of connection x.

An application program can also form a logical connection with one or more other applications and, in fact, can have several connections with another application program simultaneously, using separate and distinct logical connections. A logical connection can, therefore, refer to either a terminal or to another application. This manual uses the term connection to cover both possibilities. Typical logical connections in the network are shown in figure 1-4.

OWNING CONSOLES

Passive devices are serviced on separate logical connections from their corresponding interactive consoles. Because of this, a mechanism is needed to associate a passive device with the console that enters controlling information for it. The mechanism used is the owning console concept.

When a passive device is defined in the network configuration file, an interactive console is identified as the owning console of the passive device. The method used identifies the console by its terminal name, as defined for the console in the network configuration file. An application program receives the name of the owning console as a parameter in the passive device's connection request, along with the terminal name of the passive device. The application program also receives the terminal name of the console as part of the console's connection request, and can therefore associate the two devices.

NETWORK ACCESS METHOD OPERATION

Figure 1-5 shows the components of NAM as it is discussed in this manual. All of the area enclosed by the dotted lines comprises the Network Access Method.

As NAM receives data from the network terminals or application programs, the data is buffered in NAM's buffers. (See section 4.) Application programs use calls to AIP procedures to request and transmit this data.

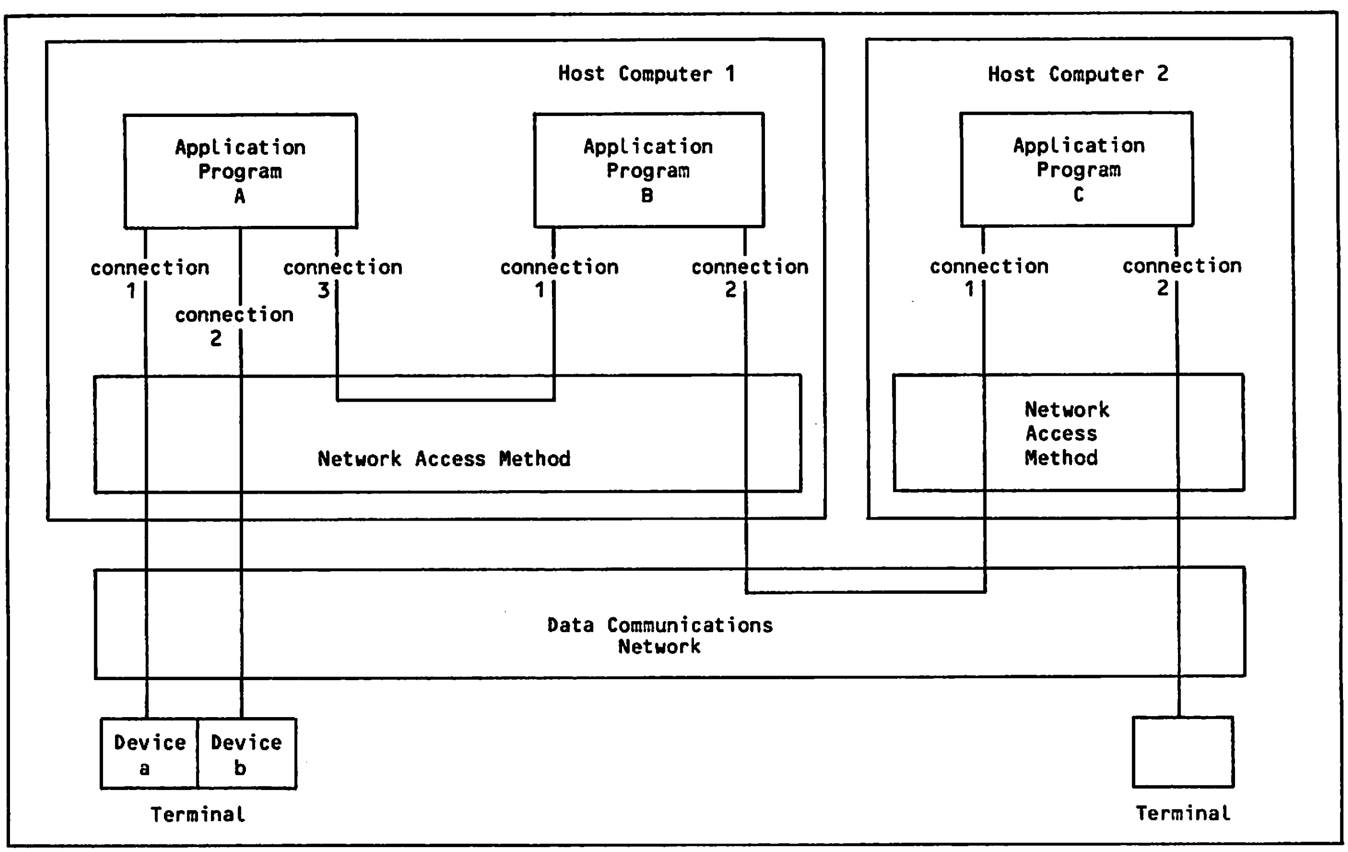
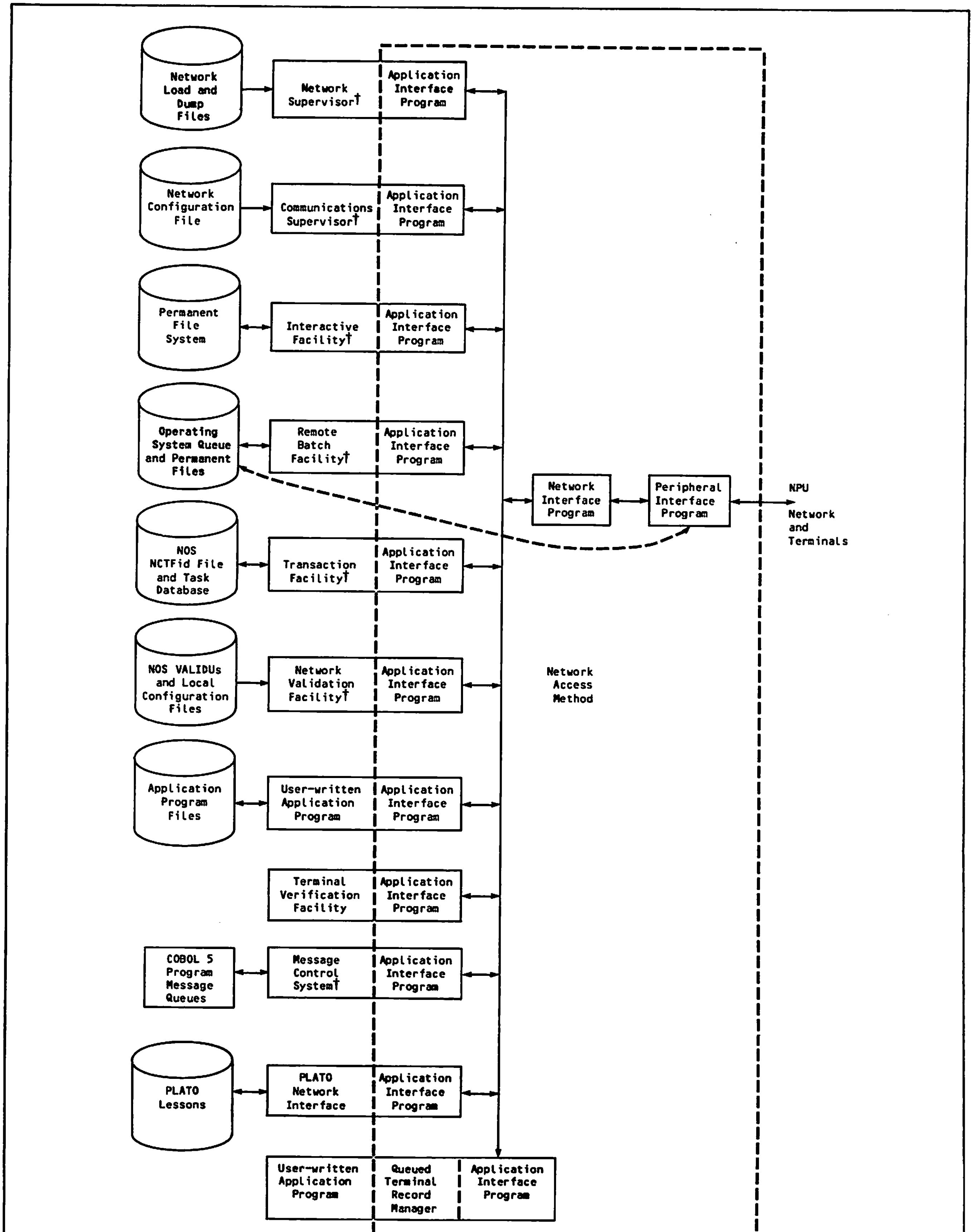


Figure 1-4. Typical Connections in the Network



†Privileged application programs; see Section 6

Figure 1-5. Network Access Method Components

Inbound data from an interactive virtual terminal or another application is placed, unmodified, in NIP's central memory buffers by PIP. These buffers form an input queue associated with the logical connection that originated the data. Data is removed from this input queue when application program AIP statements request input from the logical connection. The data can be translated and converted by NIP from ASCII to display code if the application program has requested such conversion; transparent data, as described in section 2, is neither edited nor translated. NIP places the translated or transparent data in a data buffer within the application program's field length. This data buffer is established and maintained by the application program.

Output for an interactive virtual terminal or another application is handled in the reverse manner. The application program calls an AIP procedure to send data on a logical connection. The data is transferred from the program's field length to an output queue within NIP's field length. From there, it is placed in one of PIP's output buffers, according to its priority as a supervisory message, low priority data, or high priority data, and to its destination. Code conversion and translation, if necessary, is done by PIP.

The files shown in figure 1-5 are maintained by code independent of NAM. Named files in the figure are discussed briefly in various portions of this manual.

APPLICATION PROGRAM CONCEPTS

NAM requires an application program to reside at a separate operating system control point. This program contains calls to the AIP routines listed in appendix D and described in sections 5 and 7. These calls can be direct, or indirect through the Queued Terminal Record Manager.

An application program begins accessing the network by calling NETON. It transmits data through the network by calling NETPUT or NETPUTF. It receives data through the network by calling NETGET, NETGETL, NETGETF, or NETGTFL.

An application program must contain buffers for transmitted or received data. These buffers can be either unified or fragmented central memory areas. One buffer can be used for all logical connections, or many unified buffers or fragments of a buffer can be used for each logical connection.

An application program sends instructions to the network software and receives operational information from the network software through supervisory messages, as described in section 3. It must contain procedures to formulate or process these messages.

An application program can contain procedures that optimize its use of central memory and the control processor. AIP routines can make the program available for rollout when the program has no data to process (NETWAIT), or allow the program to perform non network processing while waiting for completion of a network processing task (NETSETP and NETCHEK).

An application program can compile statistics about its functioning (NETSTC) that can be examined for application tuning. It can also cause trace dumps of its network traffic (NETDBG). The trace file generated can be dynamically disposed for storage, processing (NETREL), and application debugging.

An application program must contain a call to NETOFF to terminate its access to the network. Application programs using the optional code controlled by NETDBG or NETSTC must also dispose of the local files created by this code. (See section 6.)

CONNECTION PROCESSING FLOW

The functions performed by NAM and other software described previously in this section can best be summarized by tracing the job processing involved for a single terminal and a single site-written application program. Figure 1-6 is a generalized version of this processing flow. Time elapses in the figure from top to bottom. Program processing begins from the left, terminal actions begin from the right. Dotted lines separate functions for each entity. When the boxes formed by solid or dotted lines are aligned, the functions of the entities involved are related. Actions for a batch device (a passive device) differ from those shown for an interactive terminal; the first two and last three terminal actions are performed internally by the Network Validation Facility for batch devices based upon login information supplied for the device's owning console.

SUPPORTED TERMINALS

The network software, and therefore an application program, can service any real terminal compatible with one of the terminal classes listed in table 1-2. Each terminal class is identified by its terminal class number, described in section 3 under Managing Logical Connections. All terminal classes are supported by the interactive virtual terminal interface. When a mnemonic appears in table 1-2, it indicates the archetype terminal supported for the given terminal class and device type.

The archetype mnemonics are not used by the application program in any form; the archetypes are described in more detail in the Network Definition Language reference manual, where they are identified by the same mnemonics. (See the preface.)

Site-modified versions of the network software can service terminals in terminal classes other than those listed. This manual applies only to support of the terminal classes defined by CDC. Content of this manual can be valid for site-defined terminal classes; CDC is not responsible for deviations from this manual attributable to support of site-defined terminal classes.

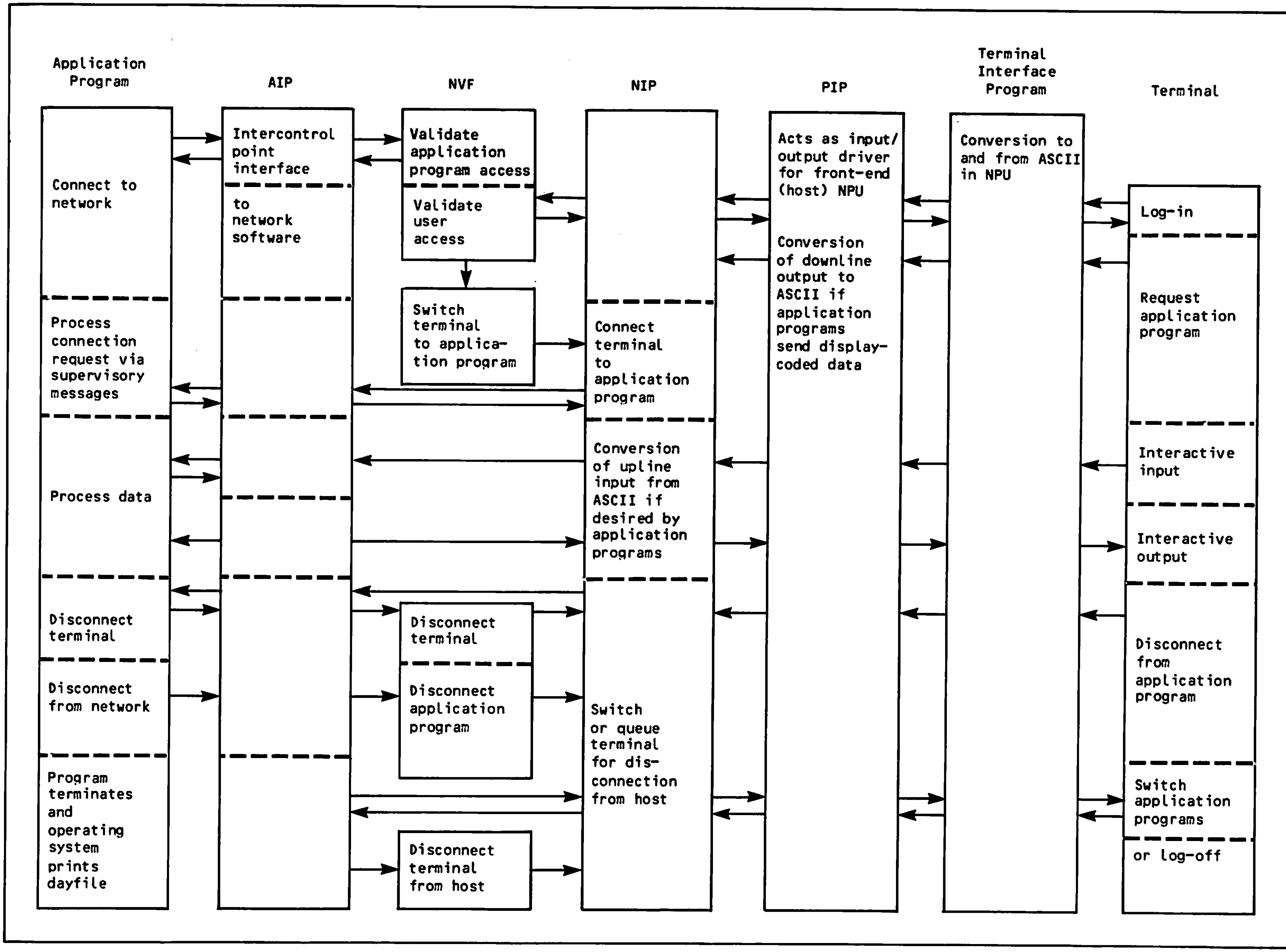


Figure 1-6. Typical Application Program Processing Flow

TABLE 1-2. SUPPORTED TERMINAL CLASSES

Line Protocol	Terminal Class	Device and Archetype Terminal Mnemonic†				
		Console	Card Reader	Line Printer	Card Punch	Plotter
Asynchronous or X.25 PAD††	1	M33				
	2	713				
	3	721				
	4†††	2741				
	5	M40				
	6	H2000				
	7	X3.64§				
	8	T4014				
HASP Bisynchronous††	9	HASP (post-print)	HASP (post-print)	HASP (post-print)	HASP (post-print)	HASP (post-print)
	14	HASP (pre-print)	HASP (pre-print)	HASP (pre-print)	HASP (pre-print)	HASP (pre-print)
Mode 4 Synchronous	10	200UT	200UT	200UT		
	11	714X		714X		
	12	711				
	13	714		714		
	15	734	200UT	200UT		
2780/3780 Bisynchronous††	16	2780	2780	2780	2780	
	17	3780	3780	3780	3780	
3270 Bisynchronous	18	3270		3270		

†A blank indicates the device type is not supported for the terminal class.

††Point-to-point configurations only. Multidrop configurations are not supported.

†††X.25 PAD does not support terminal class 4.

§Terminal such as VT100 that follows ANSI standard X3.64.

INFORMATION PROTOCOLS

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.

Information flow within a host computer is defined from the viewpoint of a network application program. Information coming to the application is said to be traveling upline; information moving away from the application 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

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.

Information exchanged between a network processing unit and a public data network use packets as the physical structure protocol. When the application communicates with a terminal or other CDC host applications, the relationship of a packet to a network block is not significant to an application programmer. Therefore, this relationship is not discussed in this section.

However, the relationship of a packet to a network block may be significant if the application is communicating with a foreign host's application. The mapping of network blocks into the X.25 protocol is discussed in the Communications Control Program Internal Maintenance Specifications.

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.

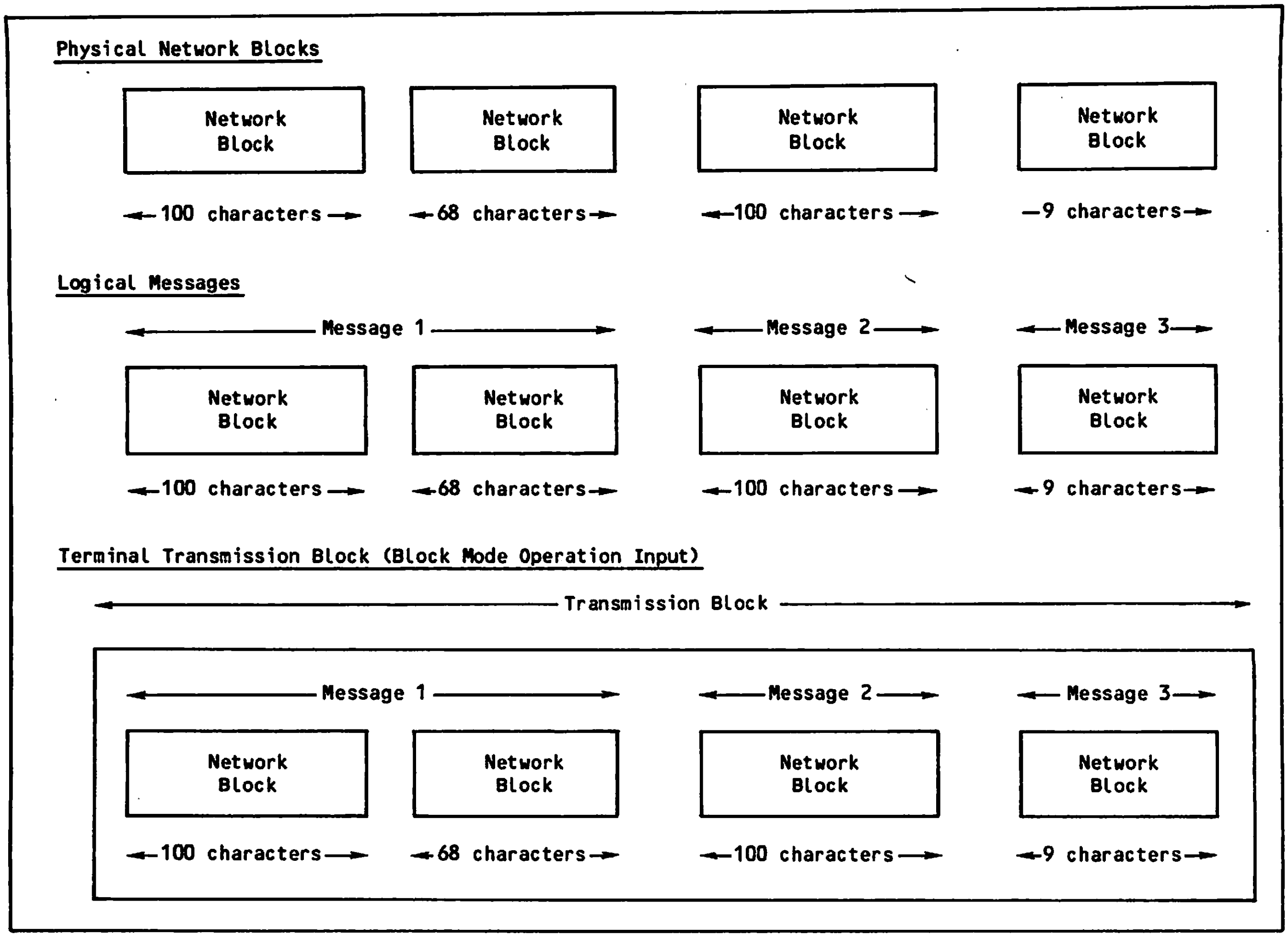


Figure 2-1. Physical and Logical Information Structures

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.

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.

Application-to-application connections have upline and downline blocks of varying sizes. The upline block size seen by one application is the downline block size used by the other application.

CDC-defined batch devices have network data block sizes that are multiples of 64 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 or application-to-application connection 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 or application connection that

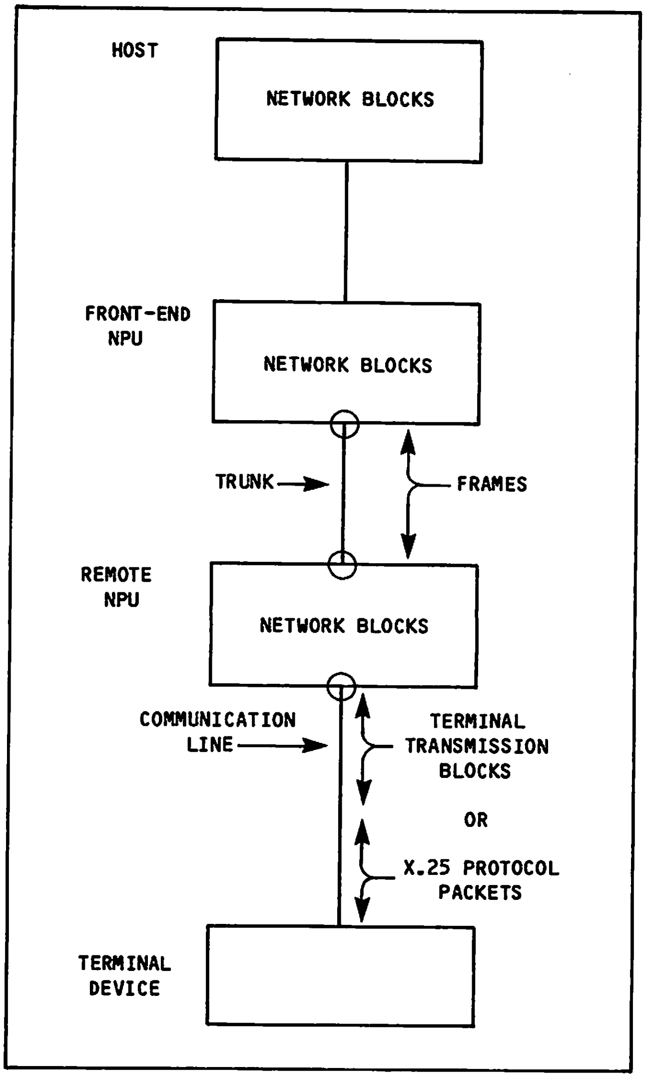


Figure 2-2. Block Reassembly Points

produces the block. Once a size is established for a connection, 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.

Application-to-application data is not split into smaller blocks before upline delivery if the data crosses a trunk line between two host nodes or if it is passed between two programs in the same host. Such data does not pass through the NPU software that prepares all other upline blocks.

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 or application connection 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.

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 and terminal transmission block size defined 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. Ignoring a recommended value can cause resource problems for the network software, particularly in the NPUs.

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 from a device.

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.

The downline terminal transmission block size is also enforced by the network software. Your application program can change the established transmission block size using a field number/field value pair, as described in section 3.

Application programs should use the downline block sizes defined 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 connection, device, you should discuss the situation with the network administrator who configures the devices being serviced. The 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. Another NDL statement parameter, the UBL parameter, establishes the upline block limit. The upline block limit determines the number of upline blocks NAM queues for your program before rejecting further input.

The upline 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 application block limit is another device or application connection 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 use of the application 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 reassembles 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.

X.25 devices (terminal classes 1 through 3 and 5 through 8), HASP and 2780/3780 devices (terminal classes 9, 14, 16, 17, and 18) always operate in line mode. 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 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. Mode 4 devices, for example, are always treated as block mode devices.

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 is initially 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. If possible, the user should change the end-of-block indicator to the code actually sent by the key. If not possible, if the code sent by the key cannot be determined, or if the key does not generate a code, then the user 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, 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).

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, 752, 756	RETURN or CARRIAGE RETURN	RETURN or CARRIAGE RETURN	SEND or CONTROL and D
3	CDC 721	NEXT	NEXT	NEXT
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	VT 100	CARRIAGE RETURN	CARRIAGE RETURN	CTRL and D
8	Tektronix 4014	RETURN	RETURN	CTRL and D
1 thru 3 5 thru 8	X.25 packet assembly/ disassembly (PAD) console device	Same as above	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	IBM 3270	ENTER	None	None
19 thru 28	Reserved for CDC use			
29 thru 31	Site-defined	Unknown	Unknown	Unknown

This option allows the terminal user to pack many logical lines into one upline network block. Each line includes the end-of-line indicator as a data character that terminates it. This is a form of line 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.

INTERACTIVE TERMINAL OUTPUT CONCEPTS

A downline message can contain no logical lines (an empty block or a transparent mode block) or 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 DEVICE DATA

Batch devices can be serviced as site-defined device types 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.

APPLICATION-TO-APPLICATION INPUT AND OUTPUT CONCEPTS

Application programs within the same host exchange data by transferring the contents of 60-bit central memory words between control points. A program can create a connection to itself and exchange data on that connection.

Application programs in different hosts exchange data by transferring the contents of 8-bit bytes through the network, as if the data were sent to or received from an interactive virtual terminal.

Application programs can exchange data only in transparent mode. Upline and downline messages are not subdivided into logical lines. Embedded codes are not used to terminate lines or network data blocks within the messages.

INFORMATION IDENTIFICATION PROTOCOLS

CDC network host software uses four general conventions 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 connection identifiers are called the application connection number and the application list number.

The following subsections describe these conventions.

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, ICMD, ICMDR, and other acronyms in some network documentation.

Blocks containing portions of qualified data messages, but not terminating those messages. These blocks have an application block type number of 6; such blocks are called QBLK blocks in other network documentation.

Blocks that terminate qualified data messages. These blocks can include physically empty blocks when such blocks convey logical information. Blocks that terminate qualified data messages have an application block type number of 7; such blocks are called QMSG blocks in other network documentation.

Qualified data can be used only on application-to-application connections. Such data has no special significance to the CYBER 170 network software. Qualified data is intended for application programs in order for such programs to communicate control information among themselves that is outside the data stream but synchronous with it. For example, user identification information (qualified data) placed before data in transferring files.

Blocks with an application block type of 6 or 7 cannot be sent or received on the logical connection between blocks with an application block type of 1 or 2. Qualified data can only be sent or received after an unqualified message ends or before an unqualified message begins.

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 through 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 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.

Upline data blocks can be truncated to fit into the existing text area. Alternatively, the application program can use a large text area for large blocks and a small text area for small blocks.

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 through 410 60-bit words or 1 through 2043 8-bit or 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), and terminal class 18 (3270 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 messages. 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 codes 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 connected to the host. This mode is complete editing. 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)

0A (the ASCII character LF)

7F (the ASCII character DEL)

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:

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

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)

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 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 follows an 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 follows an end-of-line or end-of-block indicator and is explicitly supporting paper tape input from the device (see 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.

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 18 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.

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, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 3NUL
	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; delay 100 milliseconds 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

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
	.	Space 1 line after output.	Does not matter	Yes or No	CR, LF	CR, LF DC3, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 3NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF
3	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, FF	EM, FF
	,	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, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 3NUL
	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

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
	*	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
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, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	ESC, G	ESC, G, DC3, 3NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	None LF	None LF

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	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, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	None	DC3, 3NUL
	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	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,[,H, ESC,[,J	ESC,[,H, ESC,[,J
	,	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, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 3NUL
	Any other ASCII character	Space 1 line before output.	Does not matter	Yes No	CR CR, LF	CR CR, LF

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	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
	.	Space 1 line after output.	Does not matter	Yes or No	CR, LF	CR, LF, DC3, 3NUL
	/	Position to start of current line after output.	Does not matter	Yes or No	CR	CR, DC3, 3NUL
	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.

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.

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, 15, and 18	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.

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 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 and 3270 terminals (terminal classes 10 through 13 and 15) require the special treatment described below.

Mode 4

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)

TABLE 2-4. EMBEDDED FORMAT CONTROL OPERATIONS FOR CONSOLES

Terminal Class	Format Control Character	General Physical Operation	Code Substituted on Output Mechanism
1 thru 3 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, 14, and 18	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

Downline

The application constructs a screen-full of protected/unprotected fields and supplies all the desired attribute characters and screen-buffer-addresses for the fields. The TIP is responsible for preceding the block of output by SYNC-characters, start-of-text, and escape-char, and attaches ETX,CRC,PAD at the end. The TIP also translates all downline data ASCII to EBCDIC and performs SYNC-fill.

A typical start of a field would be:

SBA	set-buffer-address x'11'	all in ASCII
BA1	buffer-address-1	
BA2	buffer-address-2	
ATT	attribute-char	

where the attribute-character determines the characteristics of the field:

- protected
- unprotected
- intensified
- numeric shift

The application is also expected to insert the cursor at a desired location.

Once transparent output is delivered to a 3270 terminal, the TIP assumes transparent input until a non-transparent downline block is delivered to the terminal.

To protect the integrity of the protocol, the TIP replaces certain downline characters by NULLs. The characters replaced are:

SOH,STX,ETX,EOT,ENQ,ACK,NAK,SYNC

Upline

Once transparent output is delivered, the TIP sends to the host all modified, unprotected fields received from the terminal including the SBA and buffer-address-chars (2) of each field. The terminal does not send the attribute characters back to the TIP.

If the incoming text is larger than one transmission block (256 characters), the TIP will send

BLK/BLK/.../MSG

so that the application can reproduce a full screen.

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, when a message ends, 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

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 may be modified as described under Changing Device Characteristics 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. These indicators have no significance to the network software.

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 or ignore parity is declared. 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 or ignore 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 or ignore parity is declared, the seventh bit is treated as part of the character data and is not changed during input or output.

APPLICATION-TO-APPLICATION CONNECTION DATA

Because application-to-application connection data is always exchanged in transparent mode, programs can exchange character data in bytes of any size. The program at both ends of the connection must interpret the data using the same byte size.

Programs within the same host can exchange 7-bit or 8-bit character data in one of three ways:

> Exchange pairs of 60-bit bytes, each containing fifteen 8-bit data bytes
>
> Exchange 8-bit data bytes packed as 8-bit bytes

Exchange 8-bit data bytes packed within 12-bit bytes

Each of these options corresponds to an application character type, as described in the next subsection. Programs in different hosts need not use the same application character type.

Programs can exchange 6-bit character data in one of two ways:

If both programs are in the same host, they can exchange 60-bit bytes, each containing 6-bit (or 6/12-bit) data bytes.

They can exchange sets of fifteen 8-bit bytes, corresponding to two central memory words per set (twenty 6-bit characters).

Figure 2-3 illustrates these possibilities. The parity bit (bit 7 of an 8-bit byte) is not altered during transmission through the network and can always be used as data.

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

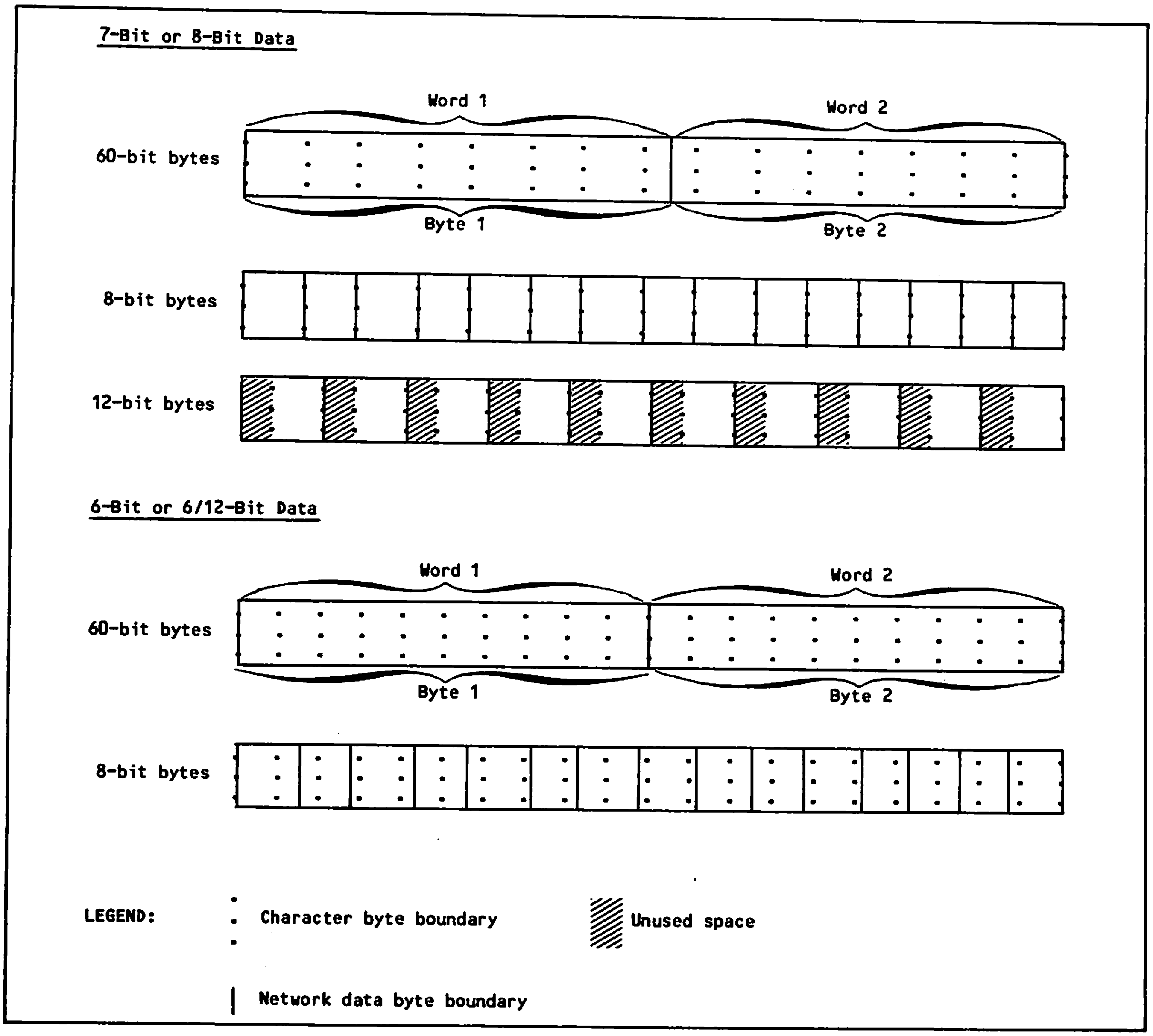


Figure 2-3. Application-to-Application Connection Data Exchanges

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 valid 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 63-character or 64-character 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. Blocks exchanged in transparent mode are identified by a field in their associated block header word.

Blocks exchanged with another application program are always exchanged in transparent mode. Transparent mode is the initial and only exchange mode for such connections. Such blocks need not have transparent mode use 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 2-4. The requirements for all header words associated with downline data blocks are described in figure 2-5.

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 63-character or 64-character 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. Blocks exchanged in transparent mode are identified by a field in their associated block header word.

Blocks exchanged with another application program are always exchanged in transparent mode. Transparent mode is the initial and only exchange mode for such connections. Such blocks need not have transparent mode use 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 2-4. The requirements for all header words associated with downline data blocks are described in figure 2-5.

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 2-6 and 2-7. 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.

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 2-6 and 2-7 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)

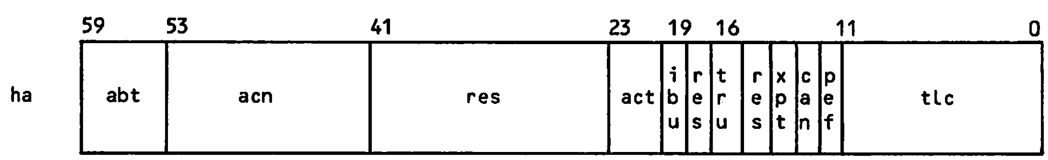
Terminate output (TO)

Break indication (BI)

Resume output (RO)

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	Transparent	Application-to-application within the same host	Binary (None)
8-bit characters in 8-bit byte	2	Normalized	Application-to-device (consoles)	7-bit ASCII (128 ASCII)
8-bit characters in 8-bit bytes	2	Transparent	Application-to-device (consoles)	Any 6-, 7-, or 8-bit (Unknown)
8-bit characters in 8-bit bytes	2	Transparent	Application-to-application	Binary (None)
8-bit characters in 12-bit bytes	3	Normalized	Application-to-device (consoles)	7-bit ASCII (128 ASCII)
8-bit characters in 12-bit bytes	3	Transparent	Application-to-device (consoles)	Any 6-, 7-, or 8-bit (Unknown)
8-bit characters in 12-bit bytes	3	Transparent	Application-to-application	Binary (None)
6-bit characters in 6-bit bytes	4	Normalized	Application-to-device (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.

=6 indicates that the associated block is one of several blocks comprising a single qualified data message, but is not the last such block.

=7 indicates that the associated block is either the last or only one comprising a qualified data message.

Values of 3 through 5 and 8 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 ≤ 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 2-4. 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 transparent mode or normalized mode data on device-to-application connections and for application-to-application connections between hosts.
=3	8-bit characters, right-justified in 12-bit bytes with zero fill, packed 5 per central memory word; this character type can be used for transparent mode or normalized mode data on device-to-application connections and for application-to-application connections.
=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 device-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.

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

- 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.

- 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).

res Reserved for CDC use.

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).

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.

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

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).

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, must 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.

act=5 thru 15 tlc is undefined.

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

†The xpt value will always be set to 0 in the upline network block if the data passes through a packet switching network. Therefore, to get consistent results, it is strongly suggested that xpt=0 be used on all application-to-application connections.

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

	59	53	41	23	19	15	14	13	12	11	0
ha	abt	acn	abn	act	0	ncp	nfe	xpt	nep	res	tlc

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.
- =6 indicates that the associated block is one of several blocks comprising a single qualified data message, but is not the last such block.
- =7 indicates that the associated block is either the last or only one comprising a qualified data message.

Values of 0, 3 through 5, and 8 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 $1 \le minacn \le acn \le maxacn \le 4095$, 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 transparent mode data or normalized mode data on device-to application connections or for application-to application connections between hosts.
- =3, 8-bit characters, right-justified in 12-bit bytes, packed 5 per central memory word; this character type can be used for transparent mode or normalized mode data on device-to-application connections, or for application-to-application connections.

Figure 2-5. Application Block Header Content for Downline Network Data Blocks (Sheet 1 of 3)

=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).

ncp No-cursor-positioning bit, indicating whether cursor positioning is to be disabled for the input operation that immediately follows this output block. If ncp is 1, no cursor positioning is to be performed for the next input operation; if ncp is 0, cursor positioning can be performed for the next operation. This bit is ignored for blocks sent on application-to-application connections and for blocks with an abt of 1 on device-to-application connections. You can access this field with the reserved symbol ABHNCP (see section 4).

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 does not determine data translation and can be 1 or 0. A value of 0 is recommended.

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 (if parity of none is used), but such data must be correctly formatted for terminal output.

act=2, if the data is for an application program, xpt does not affect data translation and can be 1 or 0. For data passing through a public data network, the receiving application will always see xpt=0. Therefore, it is strongly recommended that a value of xpt=0 be used by the sender.

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 (if parity of none is used), but such data must be correctly formatted for terminal output.

act=3, if the data is for an application program, xpt does not affect data translation and can be 1 or 0. For data passing through a public data network, the receiving application will always see xpt=0. Therefore, it is strongly recommended that a value of xpt=0 be used by the sender.

act=4, xpt value does not determine data translation and can be 1 or 0. A value of 0 is recommended.

Figure 2-5. Application Block Header Content for Downline Network Data Blocks (Sheet 2 of 3) act= other — xpt is not defined.

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

nep — No-echoplexing bit, indicating whether the next logical line of nontransparent input data should not be echoplexed. If nep is 1 and the NPU is echoing characters back to the terminal (Y value of EP command, described in NAM Version 1/CCP Version 3 Terminal Interfaces reference manual), the NPU does not echo the next logical line from the console. If nep is 0 and the NPU is echoing characters (Y value of EP command), the NPU does echo the next logical line of input. This bit is ignored for blocks sent on application-to-application connections and for blocks with an abt of 1 on device-to-application connections. You can access this field with the reserved symbol ABHNEP (see section 4).

res — Reserved for CDC use. Reserved fields contain zero.

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.

act=5 thru 15 — tlc is not defined.

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 2-5. Application Block Header Content for Downline Network Data Blocks (Sheet 3 of 3)

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 2-6 and 2-7 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)

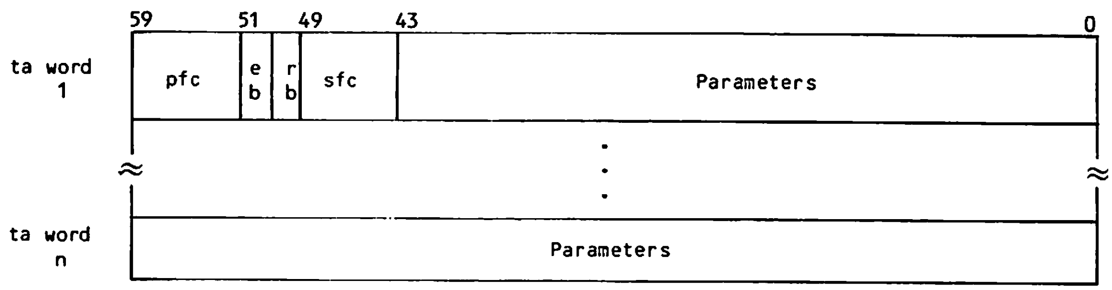
Request terminal characteristics (RTC)

Define single terminal characteristic (DEF)

Define multiple terminal characteristics (TCD)

Downline CCP terminal multiple characteristics definition (CHAR)

Define CDCNET terminal characteristics (CTD)



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
bi†	BI	312	CA	202
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
ro†	RO	313	CB	203
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).

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:

Figure 2-6. Supervisory Message General Content, Asynchronous Messages and Synchronous Messages of Application Character Type 2 (Sheet 1 of 2)

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
ccd†	CTRL	CCD	14	0C	12
ctd†	CTRL	CTD	02	02	02
def†	CTRL	DEF	04	04	04
char†	CTRL	CHAR	10	08	08
rcc†	CTRL	RCC	13	0B	11
rtc†	CTRL	RTC	11	09	09
tcd†	CTRL	TCD	12	0A	10
cict	DC	CICT	00	00	00
stmr	DC	STMR	02	02	02
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
brk	HOP	BRK	00	00	00
cmd	HOP	CMD	01	01	01
trace	HOP	TRACE	02	02	02
du	HOP	DU	03	03	03
ig	HOP	IG	04	04	04
start	HOP	START	05	05	05
endd	HOP	ENDD	06	06	06
notr	HOP	NOTR	07	07	07
rs	HOP	RS	10	08	08
dis	HOP	DIS	11	09	09
lg	HOP	LG	12	0A	10
alt	HOP	ALT	13	0B	11
page	HOP	PAGE	14	0C	12
rel	HOP	REL	15	0D	13
db	HOP	DB	16	0E	14
de	HOP	DE	17	0F	15
day	HOP	DAY	20	10	16
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 or BI or RO	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 ≤ 410. Parameters are defined in the descriptions of the specific messages in section 3.

---

†Synchronous supervisory message fields.

Figure 2-6. Supervisory Message General Content, Asynchronous Messages and Synchronous Messages of Application Character Type 2 (Sheet 2 of 2)

Functions routed between NAM and the application program are represented in figures 2-6 and 2-7 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)

Break indication (BI)

Resume output (RO)

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 2-6 and 2-7 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. 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. 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 2-8. The requirements for all header words associated with outgoing supervisory messages are described in figure 2-9.

	59	53	41	23	19			16	11	0
ha	abt	adr	Reserved for use by CDC	act	ibu	0	tru	re	tlc	

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 2-8. 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 2-8. Application Block Header Content for Upline Supervisory Messages (Sheet 2 of 2)

	59	53	41	23	19	11 … 0
ha	abt	adr	abn	act	0	tlc

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 2-9. 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 MNEMONICS

Figure 2-6 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.

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 2-6 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 2-6 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

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 blocks
- Truncating blocks
- Managing terminal characteristics
- Host operator communication
- Host shutdown
- Error reporting

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 host software sending or receiving the message, regardless of the software module actually involved.

TABLE 3-1. LEGAL SUPERVISORY MESSAGES

Message Mnemonic	Message Meaning	Type	Block Header Fields	Figure Number Defining Message
BI/MARK/R	Break-indication-marker request	Upline synchronous	acn $\neq$ 0 act = 2,3 tlc = 2	3-32
CON/ACRQ/A	Rejection of application-to-application connection request	Upline asynchronous	acn = 0 act = 1 tlc = 2	3-13
CON/ACRQ/R	Application-to-application connection request	Downline asynchronous	acn = 0 act = 1 tlc = 2	3-12
CON/CB/R	Connection broken	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-8
CON/END/N	All connection processing completed	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-10
CON/END/R	End all connection processing	Downline asynchronous	acn = 0 act = 1 tlc $\geq$ 2	3-9
CON/REQ/A	Connection rejected	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-5
CON/REQ/N	Connection accepted	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-4
CON/REQ/R	Connection requested	Upline asynchronous	acn = 0 act = 1 tlc $\geq$ 6	3-3, 3-14
CTRL/CHAR/A	No terminal characteristics changed	Upline synchronous	acn $\neq$ 0 act = 2, 3 tlc = 4	3-49
CTRL/CHAR/N	Multiple terminal characteristics defined	Upline synchronous	acn $\neq$ 0 act = 2, 3 tlc = 2	3-50
CTRL/CHAR/R	Define multiple terminal characteristics	Downline synchronous	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 2	3-48
CTRL/DEF/R	Redefine terminal characteristic	Downline synchronous	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 2	3-47
CTRL/RTC/A	Bad value in request terminal characteristics supervisory message	Upline synchronous	acn $\neq$ 0 act = 2, 3 tlc = 4	3-52
CTRL/RTC/R	Request current value of terminal characteristics	Downline synchronous	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 2	3-51
CTRL/TCD/R	Terminal characteristics definitions	Upline synchronous	acn $\neq$ 0 act = 2, 3 tlc $\geq$ 2	3-53

Message Mnemonic	Message Meaning	Type	Block Header Fields	Figure Number Defining Message
DC/CICT/R	Change application character type of connection input	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-42
DC/TRU/R	Truncate upline block	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-44
ERR/LGL/R	Logical error	Upline asynchronous	acn = 0 act = 1 tlc $\geq$ 3	3-65
FC/ACK/R	Output block delivered	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-25
FC/BRK/R	Connection processing interrupted by break	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-28
FC/INACT/R	Connection inactive	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-16
FC/INIT/N	Application ready for connection processing (connection initial-ized)	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-7
FC/INIT/R	NAM ready for connection process-ing (connection initialized)	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-6
FC/NAK/R	Output block not delivered	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-26
FC/RST/R	Reset connection	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-29
HOP/DB/R	Activate debug code	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-55
HOP/DE/R	Turn off debug code	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-56
HOP/DU/R	Dump field length	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-57
HOP/NOTR/R	Turn off AIP tracing	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-59
HOP/REL/R	Release debug log file	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-60
HOP/RS/R	Restart statistics gathering	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-61

TABLE 3-1. LEGAL SUPERVISORY MESSAGES (Contd)

Message Mnemonic	Message Meaning	Type	Block Header Fields	Figure Number Defining Message
HOP/TRACE/R	Turn on AIP tracing	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-58
INTR/APP/R	Application interrupt request	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-35
INTR/RSP/R	Interrupt response	Downline or upline asynchronous	acn = 0 act = 1 tlc = 1	3-33, 3-36
INTR/USR/R	User interrupt or user interrupt request	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-31, 3-39
LST/FDX/R	Turn on full duplex operation for connections in list	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-24
LST/HDX/R	Turn on half duplex operation for connections in list	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-23
LST/OFF/R	Turn list processing for connection off	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-20
LST/ON/R	Turn list processing for connection on	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-21
LST/SWH/R	Switch application list number of connection	Downline asynchronous	acn = 0 act = 1 tlc = 1	3-22
RO/MARK/R	Resume output marker	Downline synchronous	acn ≠ 0, act = 2,3 tlc = 2	3-34
SHUT/INSD/R	Network shut-down in progress	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-63
TCH/TCHAR/R	Terminal characteristics redefined	Upline asynchronous	acn = 0 act = 1 tlc = 1	3-46
TO/MARK/R	Terminate output marker	Downline synchronous	acn ≠ 0 act = 2, 3 tlc = 2	3-37

Application		NAM	Message
◄────────────			CON/REQ/R
────────────►			CON/REQ/N
◄────────────			FC/INIT/R
────────────►			FC/INIT/N

The application program can now send and receive messages over the logical connection.

Application		NAM	Message
◄────────────			CON/REQ/R
────────────►			CON/REQ/A

The application program has rejected the logical connection.

Application		NAM	Message
◄────────────			CON/REQ/R
────────────►			CON/REQ/N
◄────────────			CON/CB/R
────────────►			CON/END/R
◄────────────			CON/END/N

Although the application program was willing to accept it, the logical connection could not be completed.

Figure 3-2. Device-to-Application Connection Supervisory Message Sequences

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 device-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.

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 blocks on a connection until connection initialization processing has been completed.

5958 54 5251 49 47 45 43 41 39 35 31 29 25 23 2120 1716 12 7 5 3 0

ta	con	0	0	req	res	acn	abl	sdt	dt	tc	res	ric	ord
	tname	0	pw	pl									
	ownert	0	sl	dbz	hw								
	res	ubz	xbz	res									
	logfam	famord											
	logname	usrind											
ahmt	res	ahpt	ahmti	ahrp	ahdb	res	ahtl	ahsl	ahcm	ahec	ahlp	ahcp	
ahds	ahds	ahfc	ahcs	ahis	ahsc	res	ahdt	ahdf	ahcc	ahms			
aawc	res	See NOS Administration Handbook											
atwd	atpar	atro	atpx	attt	attc	atis	res	accd	acmd				

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

con
Primary function code 63_{16}. You can access this field 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. You can access this field 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 contain 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.) You can access this field 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 device involved in the logical connection when the device is described in the network configuration file. This field has the range 1 $\leq$ abl $\leq$ 7. You can access this field with the reserved symbol CONABL, as described in section 4.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 1 of 6)

sdt	Subdevice type.		
	If dt=1 or 12 through 15 (card reader or a site-defined device), 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
		2 thru 11	Reserved for CDC use
		12 thru 15	Reserved for installation use
	If dt=2 or 12 through 15 (line printer or a site-defined device), 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
		3 thru 11	Reserved for CDC use
		12 thru 15	Reserved for installation use
	If dt=4 or 12 through 15 (plotter or a site-defined device), this field can have the values:		
		0	Instructions must be packed in 6-bit bytes
		1	Instructions must be packed in 8-bit bytes
		2 thru 11	Reserved for CDC use
		12 thru 15	Reserved for installation use
dt	Device type of the terminal device. This field can have the values:		
		0	Console (interactive terminal)
		1	Card reader; your program should reject connections with this device type
		2	Line printer; your program should reject connections with this device type
		3	Card punch; your program should reject connections with this device type
		4	Plotter; your program should reject connections with this device type
		5 thru 11	Reserved for CDC use
		12 thru 15	Reserved for installation use

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 2 of 6)

Devices with a device type of zero can be serviced as interactive virtual terminals. Devices with device types of 1 through 4 must be serviced as batch devices. You can access this field with the reserved symbol CONDT, as described in section 4. Applications other than RBF are only allowed to do input/output on batch devices if the devices are of types 0 or 12 through 15.

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 the Terminal Interfaces reference manual. The terminal class field can have the values:

0	Reserved for CDC use.
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, 752, or 756.
3	Archetype terminal for the class is a CDC 721.
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 VT100 (ANSI X3.64 standard).
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.
18	Archetype terminal for the class is an IBM 3270.
19 thru 27	Reserved for CDC use.
28 thru 31	Reserved for installation use.

You can access this field with the reserved symbol CONT, as described in section 4.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 3 of 6)

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 should 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. You can access this field 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 part of the same terminal. This field can have the value: 0 All interactive consoles 1 thru 7 Batch devices The device ordinal is assigned to the device when the device is defined in the network configuration file. You can access this field 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. You can access this field with the reserved symbol CONTNM, as described in section 4.
pw	If the device is a console, 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 consoles is discussed in the Terminal Interfaces reference manual. You can access this field with the reserved symbol CONPW, as described in section 4. The pw value can be assigned in the network configuration file or the user can set console pw from the terminal. Default value depends on terminal class.
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 the Terminal Interfaces reference manual. This field can have the values 0 or $8 \leq pl \leq 255$ for interactive consoles, but is always 60 for batch devices. You can access this field with the reserved symbol CONPL, as described in section 4.
ownert	Terminal device name of the owning console (for batch devices only). For batch devices, this field contains one to seven 6-bit display code characters, left-justified with blank fill; for console devices, this field is zero. You can access this field with the reserved symbol CONOWNR, as described in section 4.
sl	Access level of the communications line in use. Access to information or resources requiring a security level higher than this value should be prohibited. This value is the AL parameter from the NDL statement defining the communication line used by the terminal. This field can have the values $0 \leq sl \leq 15$. You can access this field with the reserved symbol CONSL, 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 $1 \leq dbz \leq 2043$. The values are advisory only. You can access this field with the reserved symbol CONDBZ, as described in section 4.
hw	The hardwired line indicator. A 0 (zero) indicates that the device is not hardwired; a 1 indicates that the device is hardwired.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 4 of 6)

ubz	Upline block size (in multiples of 100 characters) for a console device. Upline block size (in PRUs) of a batch device. Console connections with an upline block size of 0 send blocks of 100 characters or blocks created when a linefeed is entered from the console. You can access this field with the reserved symbol CONUBZ, as described in section 4.
xbz	Transmission block size (in characters) of the device. This is the number of characters in an output transmission block that CCP sends to the terminal. You can access this field with 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. You can access this field with the reserved symbol CONFAM, as described in section 4.
famord	The NOS family ordinal corresponding to the logfam field contents. You can access this field with 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. You can access this field with the reserved symbol CONUSE, as described in section 4.
usrind	The NOS user index corresponding to the logname field contents. You can access this field with the reserved symbol CONUI, as described in section 4.
ahmt	User validation control word defined in the NOS validation file. You can access this word with the reserved symbol CONAHMT, as described in section 4. The NOS Administration Handbook section on the MODVAL command explains the use of the fields in this word.
ahpt	Index value of allowed units plotted per file for the connection's user name. See NOS MODVAL PT parameter.
ahmti	Index value of allowed magnetic tapes for the connection's user name. See NOS MODVAL MT parameter.
ahrp	Index value of allowed removable packs for the connection's user name. See NOS MODVAL RP parameter.
ahdb	Index value of allowed deferred batch jobs for the connection's user name. See NOS MODVAL DB parameter.
ahtl	Index value of central processor time limit per job step for the connection's user name. See NOS MODVAL TL parameter.
ahsl	Index value of system resource unit limit for the connection's user name. See NOS MODVAL JL parameter.
ahcm	Index value of allowed central memory field length for the connection's user name. See NOS MODVAL CM parameter.
ahec	Index value of allowed extended central storage field length for the connection's user name. See NOS MODVAL EC parameter.
ahlp	Index value of allowed lines printed per file for the connection's user name. See NOS MODVAL LP parameter.
ahcp	Index value of allowed cards punched per file for the connection's user name. See NOS MODVAL CP parameter.
ahds	User validation control word defined in the NOS validation file. You can access this word with the reserved symbol CONAHDS, as described in section 4. The NOS Administration Handbook section on the MODVAL command explains the use of the fields in this word.
ahdsi	Index value of allowed direct access file size for the connection's user name. See NOS MODVAL DS parameter.
ahfc	Index value of allowed maximum number of permanent files in catalog for the connection's user name. See NOS MODVAL FC parameter.
ahcs	Index value of allowed maximum total indirect access file storage space for the connection's user name. See NOS MODVAL CS parameter.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 5 of 6)

ahis	Index value of allowed indirect access file size for the connection's user name. See NOS MODVAL IS parameter.
ahsc	Allowed security count for the connection's user name. See NOS MODVAL SC parameter.
ahdt	Allowed number of detached jobs for the connection's user name. See NOS MODVAL DT parameter.
ahdf	Allowed number of calls per job to the COMPASS MSG macro for dayfile entries under the connection's user name. See NOS MODVAL DF parameter.
ahcc	Allowed number of NOS commands per job for the connection's user name. See NOS MODVAL CC parameter.
ahms	Allowed number of mass storage physical record units per job for the connection's user name. See NOS MODVAL MS parameter.
aawc	User validation control word defined in the NOS validation file. You can access this field with the reserved symbol CONAAWC as described in section 4. The NOS Administration Handbook section on the MODVAL command (AW parameter) explains the use of the fields in this word. This word contains permission bits for the connection's user name. A set bit indicates that the user name is allowed that permission.
atwd(atpa)	User validation control word defined in the NOS validation file. You can access this word with the reserved symbol CONATWD, as described in section 4. The NOS Administration Handbook section on the MODVAL command explains the use of the fields in this word.
atpar	Terminal parity associated with the connection's user name (0 means that PA command is assumed to require value of E; 1 means that PA command is assumed to require value of O). See NOS MODVAL PA parameter.
atro	Number of idle characters associated with the connection's user name. See NOS MODVAL RO parameter.
atpx	Transmission mode (0 means that EP command is assumed to require value of N; 1 means that EP command is assumed to require value of Y). See NOS MODVAL PX parameter.
attt	Terminal type associated with the connection's user name. See NOS MODVAL TT parameter. One of the following: **Bit** / **Type** 52 — Teletypewriter compatible terminal, using ASCII codes 51 — Block mode terminal, using ASCII codes 50 — CDC-713-compatible terminal 49 and 48 — Reserved for CDC use
attc	Character set associated with the connection's user name (0 means the NOS NORMAL mode 6-bit display code set is assumed to be used in permanent files accessed through the Interactive Facility; 1 means the NOS ASCII mode 6/12-bit display code set is assumed to be used in permanent files accessed through the Interactive Facility). See NOS MODVAL TC parameter.
atis	Initial Interactive Facility subsystem associated with the connection's user name. See NOS MODVAL IS parameter. One of the following: **Bit** / **Subsystem** 46 — BASIC 45 — BATCH 44 — EXECUTE 43 — FORTRAN 42 — FTNTS If no bit is set, the NULL subsystem is used; if all bits are set, the ACCESS subsystem is used.
accd	Date user name was created, in the format yymmdd.
acmd	Date user name permissions were last changed, in the format yymmdd.
awsi	The user validation control word. It is defined in the NOS validation file.

Figure 3-3. Connection-Request (CON/REQ/R) Supervisory Message Format, Device-to-Application Connections (Sheet 6 of 6)

	59	51	49	43	35	23	11	9	5	0
ta	con	0	1	req	unused	acn	unused	nxp	sct	act	aln

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}. You can access this field 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. You can access this field 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. The value placed in this field must be the value used in the CON/REQ/R message to which this message is a response. You can access this field with the reserved symbol CONACN, as described in section 4.

nxp — No transparent input allowed flag. This field can have the values:

- 0 — Deliver network data blocks when the xpt field in the accompanying block header word is 1
- 1 — Discard network data blocks when the xpt field in the accompanying block header word is 1

The change-input-character-type supervisory message, described later in this section, permits an application to change to or from allowing transparent mode terminal device input. If transparent input is not allowed any transparent input from a terminal device destined for the application will be discarded. You can access this field 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 should be used
- 1 — Application character type 3 should be used

Indicates the input character type required by the application program for synchronous supervisory messages. 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. You can access this field with the reserved symbol DCSCT, as described in section 4.

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

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:

0	Reserved for CDC use.
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 bytes 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 bytes 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. Cannot be used for application-to-application connections or connections with batch devices.
5 thru 11	Reserved for CDC use.
12 thru 255	Reserved for site-defined use.

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. You can access this field with the reserved symbol CONACT, as described in section 4.

aln	Application list number assigned by the application program to this logical connection; $0 \le \text{aln} \le 63$. You can access this field with the reserved symbol CONALN, as described in section 4.

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

	59	51	49	43	35	23	0
ta	con	1	0	req	rc	acn	unused

ta	Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.
con	Primary function code 63_{16}. You can access this field 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. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol CONACN, as described in section 4.

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

	59	51		49	43	35	23	0
ta	fc	0	0	init	unused	acn	unused	

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

fc Primary function code 83_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol FCACN, as described in section 4.

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

	59	51		49	43	35	23	0
ta	fc	0	1	init	unused	acn	unused	

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

fc Primary function code 83_{16}. You can access this field 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. You can access this field 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 FC/INIT/R message to which this message is a response. You can access this field with the reserved symbol FCACN, as described in section 4.

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

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.

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 device-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 are not already connected to the maximum number of application programs permitted.

	59	51		49	43	35	23	0
ta	con	0	0	cb	rc	acn	unused	

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

con — Primary function code 63_{16}. You can access this field 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. You can access this field 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:

- 0 — Reserved for CDC use.
- 1 — Communication has been lost with the element at the other end of the logical connection. If the element is an application program, it failed, was shutdown, or ended the connection; if the element is a device, the line has disconnected or the device failed.
- 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 the connection cannot be initialized, or if the NOP disabled the communication line used by the connection.
- 3 thru 255 — Reserved for CDC use.

You can access this field 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. You can access this field with the reserved symbol CONACN, as described in section 4.

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

	59	51	49	43	35	23	17	0
ta	con	0	0	end	0	acn	unused	
	aname						unused	

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

con — Primary function code 63_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol CONACN, as described in section 4.

aname — Name of next application, one to seven 6-bit display coded characters consisting of letters or digits only with a leading alphabetic character, left-justified and blank filled within the field. This field is 0 for application-to-application connections. For device-to-application connections, 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 device or disconnects the device from the host. 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 (VALIDUs).

Figure 3-9. End-Connection (CON/END/R) Supervisory Message Format

	59	51	49	43	35	23	0
ta	con	0	1	end	unused	acn	unused

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

con — Primary function code 63_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol CONACN, as described in section 4.

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

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 host software sending or receiving the message, regardless of the software module actually involved.

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 device-to-application connections. If the latter occurs, both application programs receive the form of the asynchronous connection-request supervisory message with the form shown in figure 3-14. Both programs may accept the connection by issuing 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.

Application 1	NAM	Application 2	Message
——→			CON/ACRQ/R
		——→	CON/REQ/R
←——			CON/REQ/R
		←——	CON/REQ/N
——→			CON/REQ/N
		——→	FC/INIT/R
←——			FC/INIT/R
		←——	FC/INIT/N
——→			FC/INIT/N

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

Application 1	NAM	Application 2	Message
——→			CON/ACRQ/R
←——			CON/ACRQ/A

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

Application 1	NAM	Application 2	Message
——→			CON/ACRQ/R
		——→	CON/REQ/R
←——			CON/REQ/R
		←——	CON/REQ/A
——→			CON/REQ/N
←——			CON/CB/R
——→			CON/END/R
←——			CON/END/N

Application program 2 rejects the logical connection.

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

	5958 55 52 49 47 43 39 35 31 27 23 17 15 7 0								
ta	con	0	0	acrq	0	lid			
	name1	name2							
	A1	0	dbl	dbz	abl	0	ubl	ubz	res
	res	res	ws	dpls	facn	cudl	res		
	res								
	res								
	facl	fac							
	⋮	⋮							
	facl	fac							
	prid								
	udata (0-124 octets)								

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

con Primary function code 63_{16}. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol ACRQ.

lid Logical Identifier. It is optional but at least one of the parameters LID/NAME2, must be specified for interhost connections.

If a logical identifier is specified, then that LID should have been previously specified in the LIDCMid file. (See NOS IHB.) If a LID is specified and NAME2 is not specified, then a physical identifier (PID) that is linked to NAM at the time of issuing the CON/ACRQ message is used as NAME2 in the OUTCALL search.

If both LID and a NAME2 parameters are specified, then NAME2 is assumed to be a PID, and must have been previously specified as a legal PID for the LID in the LIDCMid file, and the PID must be linked to NAM at the time of issuing a CON/ACRQ message.

Note: For NAM to be able to detect that a PID is linked to NAM, the PID must have been previously used as a PID=xxx parameter in an OUTCALL statement in the LCF previously created by NDL.

name1 Outcall Identifier, 1-7 alphanumeric characters with a leading alpha, left justified and blank-filled. This parameter is used to uniquely identify the appropriate OUTCALL definition that establishes a connection to another application.

Figure 3-12. Request-Application-Connection (CON/ACRQ/R) Supervisory Message Format (Sheet 1 of 3)

name2	Outcall Identifier, 1-3 alphanumeric characters, left justified and blank-filled. This parameter is optional (see LID parameter); when explicitly specified in the CON/ACRQ message, or when implied by the LID, together with NAME1, it is used to select the appropriate OUTCALL definition from the collection of outcall definitions as previously specified by the Network Definition Language OUTCALL statement during the creation of the Local Configuration File (LCF). Thus the combination of NAME1 and NAME2 (implicit or explicit) must appear as NAME1 and NAME2 or PID on an OUTCALL statement. For intra-host connections, both the LID and the PID may be zero. If the application supplies its own outcall block, then the explicit or implicit PID must have appeared on a PID parameter in the OUTCALL statement of a previously created LCF. The parameters that follow (A1 through udata) are application supplied OUTCALL parameters. An application may supply its own OUTCALL parameters if it is a privileged application (has an SSJ= entry point, or a non-zero SSID). In this case, these parameters do not need to appear in the OUTCALL statement in the LCF.
A1	Flag indicating priority. 0 = No 1 = Yes
dbl	Downline block limit. Downline blocks that can be outstanding between the host computer (i.e., NAM) and the other end of this logical connection. The value chosen determines how many blocks of data the NPU queues from the total number of outstanding blocks (APL parameter value) of the size specified by the dbz. This parameter is optional and has a range of 1 ≤ dbl ≤ 7.
dbz	Downline block size. The recommended maximum number of 8-bit character bytes in any network data block sent on the connection. This field can have values 0 ≤ dbz ≤ 20, where 0, 1 both indicate 100-byte blocks.
abl	Application block limit. Specifies 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 field has the range 1 ≤ abl ≤ 7. You can access this field with the reserved symbol CONABL, as described in section 4.
ubl	Upline block limit. This parameter specifies the maximum number (1 ≤ upblim ≤ 31) of blocks that the NPU can have outstanding (unacknowledged) to the calling host. This parameter is meaningful only for X.25 connections.
ubz	Upline block size. This parameter specifies the maximum number (1 ≤ upsize ≤ 2000) of bytes that the NPU can send to the calling host in a block. This parameter is only used for X.25 links.
ws	Send window size. (Applicable on Public Data Network A-A connections only. Ignored on other A-A connections.)
dpls	Send data packet length. (Applicable on Public Data Network A-A connections only. Ignored on other A-A connections.)
facn	Number of facility groups. (Applicable to Public Data Network A-A connections only.)
cudl	Length of call user data (in octets).
facl	Facility codes length, within the CM word. (Applicable to Public Data Network A-A connections only.)
fac	Facility codes. (Applicable to Public Data Network A-A connections only.)
prid	Protocol ID. (Applicable to Public Data Network A-A connections only.) 1-8 hexadecimal digits, left justified, zero filled. If CUDL ≠ 0, then only the first 6 hexadecimal digits will be passed on to the PDN, the last two hexadecimal digits will be zeroed.

Figure 3-12. Request-Application-Connection (CON/ACRQ/R) Supervisory Message Format (Sheet 2 of 3)

udata	Call user data. If the destination host is a NOS system running network products, the first 12† octets must be of the form SSS DD AAAAAAA, where:

SSS	is the 3 ASCII character equivalent of the SNODE (sendng node number) value, right justified, zero-character filled.
DD	is the 2 ASCII character string equivalent of the DHOST (destination host number) value, right justified, zero-character filled.
AAAAAAA	is the 7 ASCII character string equivalent of the called applica- tion's application name, left justified, blank-character filled.

The remainder of the UDATA filled (0-112 octets) will be passed to the called application as user data.

At any rate, the called host/application if accessed through a public data network must be able to support the Fast Select Facility, if more than 12 octets of information are specified.

Note: For applications accessing foreign hosts through a public data network the 4 octets of the PRID field and the (up to) 124 octets of the UDATA field are combined into the (up to) 128 octets of used data as defined by the CCITT recommendation for X.25 networks.

---

†An octet is 8 bits of information.

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

	59	51		49	43	35	17 0
ta	con	1	0	acrq	rc	abn	reserved
	name1						name2

ta	Symbolic address of the application program's text area receiving this asynchronous supervisory message.
con	Primary function code 63_{16}. You can access this field 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. You can access this field 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. The field is actually made up of two 4 bit subfields, rc1 and rc2. The rc1 field comprises bits 40-43 and the rc2 field comprises bits 36-39.

The rc2 field is used so that the application can determine what action to take when it receives a CON/ACRQ/A message and it provides some general information about the source of the trouble. This field can have the following values:

1 = Critical error in call request detected by source host (only LID/PID/NDL configuration changes or application code changes would solve the problem).

2 = Critical error in call request detected by destination host.

3 = Source host temporarily cannot make the connection (resources are currently not available, but they might become available without operator intervention).

4 = Destination host temporarily cannot make the connection.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format (Sheet 1 of 4)

5 = Source host cannot make the connection for an indefinite period of time (resources can be made available by operator intervention such as enabling a LID/PID, network element, or bringing up a system or subsystem).

6 = Destination host cannot make the connection for an indefinite period of time.

Thus if rc2 = 1 or 2, the application would not try establishing the connection again, it would notify the user and/or operator that the connection is not possible.

If rc2 = 3 or 4 then the application can retry the CON/ACRQ message after a shorter period of time, and if rc2 = 5 or 6 then it will retry the CON/ACRQ after a somewhat longer period of time.

The rc1 field is used in combination with the rc2 field to uniquely identify the exact source of the trouble, so that the user/operator can take the appropriate action to fix the problem. The full 8 bit reason code field can therefore have the following values:

2 = Network error detected by destination host. Contact system analyst at destination host.

4 = Connection number conflict between source and destination host. Retry connection request.

17 = Illegal LID/PID combination was specified. Correct LID/PID in OUTCALL block.

18 = Called application is not defined in system record (CONTNAP) at destination host. Contact system analyst.

19 = Network Validation Facility (NVF) temporarily cannot process connection request. Retry later.

20 = Called application cannot accept any more connections and another copy of the application cannot be started up. Retry later.

22 = Called application is not running and cannot be started automatically. Contact system analyst to start up called application.

33 = Calling application is not privileged, i.e., it is not allowed to issue OUTCALLS. Contact system analyst to make the application a privileged application in the LCF.

34 = OUTCALL block has facility parameters greater than 4 octets in length. Correct the OUTCALL block.

35 = NAM temporarily cannot complete the connection request because the (logical) link to the destination host is not available. Retry later.

37 = Specified PID is valid but is currently not available. Retry later.

38 = Called application is disabled. Contact system analyst to enable the application.

49 = Application specified its own OUTCALL parameters but there was no corresponding OUTCALL entry in the LCF for the same PID. Correct the OUTCALL parameters in the CON/ACRQ/A.

50 = OUTCALL block had user parameters greater than 124 octets in length. Correct the OUTCALL block.

53 = Source host is not allowing any new connections because it is in idle or disabled state. Retry later.

54 = Destination host is not allowing any new connections because it is in idle or disabled state. Retry later.

65 = Application specified its own OUTCALL parameters but there was no matching OUTCALL entry in the LCF. Correct the OUTCALL parameters in the CON/ACRQ/R.

66 = Destination host could not find a matching INCALL block in its LCF. Correct the OUTCALL block.

81 = Calling application has already reached its maximum number of allowed connections. Retry later.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format (Sheet 2 of 4)

82 = Name of application specified in CON/ACRQ is invalid. Correct the application.

97 = Retry limit has been reached for calling application. No more application to application connection requests (CON/ACRQ/R) should be issued. The reason codes for the previous CON/ACRQ/A should be analyzed.

98 = Destination host could not find a matching INCALL block in the LCF with a matching facility code. Correct the facility code in the OUTCALL block.

100 = Network Validation Facility (NVF) in the destination host has not netted on yet. Retry later.

114 = Application requested Fast select but matching INCALL block in LCF at the destination host does not have Fast select specified. Correct the OUTCALL block to not select Fast select.

129 = No X25 TIP in NPU at source host. Contact system analyst to rebuild CCP with X25 TIP.

130 = Error in incoming call packet header. Contact system analyst about possible PSN problem.

132 = Unknown packet from remote, i.e., the packet received is not a call accepted or call connected. This is assumed to be caused by a call collision. Retry later.

133 = No available logical channel at source host, i.e., active number of SVCs are greater than enabled SVCs. Contact the system analyst about enabling additional SVCs.

134 = No available logical channel at destination host, i.e., active number of SVCs are greater than enabled SVCs. Contact the system analyst at the destination host to enable some more SVCs.

145 = X25 subtip not available in NPU at source host. Contact system analyst for rebuilding CCP.

146 = X25 subtip not available in NPU at destination host. Contact system analyst at destination site for rebuilding CCP.

147 = NPU at source host temporarily has no buffer space to support the connection. Retry later.

148 = NPU at destination host temporarily has no buffer space to support the connection. Retry later.

161 = Problem detected by X25 network at local host. PSN CCC=13. Local procedure error. Clear problem with PSN administration.

162 = Remote host not known. Correct DD field in UDATA in OUTCALL entry in the LCF or in the CON/ACRQ/R message.

163 = No connection available, i.e., all SVCs (outside lines) have been used. Retry later.

164 = Problem detected by X25 network at destination host. PSN CCC=1. Number at destination host is busy. Retry later.

165 = X25 line is down at source host. Retry later.

166 = X25 line is down at destination host. Retry later.

178 = Unknown subtip connection; i.e., the PRID field is not C0 (PAD) or C1 (A-A). Fix the PRID field in the OUTCALL entry in the LCF or in the CON/ACRQ/R message.

180 = Problem detected by X25 network. PSN CCC=5. PSN congestion. Retry later.

182 = CCP cannot complete the connection because the (logical) link at the destination host is not up (enabled). The system analyst should be contacted to enable the logical link.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format (Sheet 3 of 4)

dpls	Send data packet length, specifying the maximum number of data octets (8-bit bytes) an X.25 packet can contain. This parameter applies only to X.25 network application-to-application connections and is ignored on other application-to-application connections. The dpls parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). This parameter does not need to appear in the OUTCALL statement in the LCF. You can access this field with the reserved symbol CONDPLS, as described in section 4.
facn	Number of facility groups. This parameter applies only to X.25 network application-to-application connections. The facn parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). In this case, the facn parameter does not need to appear in the OUTCALL statement in the LCF. You can access this field with the reserved symbol CONFACN, as described in section 4.
cudl	Length of user data (in octets). The cudl parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). This parameter does not need to appear in the OUTCALL statement in the LCF. You can access this field with the reserved symbol CONAUDL, as described in section 4.
facl	Facility code length, specifying the length of a facility field within the central memory word. This parameter applies only to X.25 network application-to-application connections. The facl parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). This parameter does not need to appear in the OUTCALL statement in the LCF.
fac	Facility code, specifying the facility code for a facility field. This parameter applies only to X.25 network application-to-application connections. The fac parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). This parameter does not need to appear in the OUTCALL statement in the LCF.
prid	The protocol identification. This parameter tells the PSN or remote node of a direct X.25 link how call user data is to be used. This parameter applies only to X.25 network application-to-application connections and must be 1 to 8 hexadecimal digits, left-justified, and zero-filled. If CUDL ≠ 0, only the first 6 hexadecimal digits are passed to the X.25 network, and the last two hexadecimal digits are zeroed. The prid parameter is an application supplied OUTCALL parameter. An application can supply its own OUTCALL parameters if it is a privileged application (SSJ= entry point, or a non-zero SSID). This parameter does not need to appear in the OUTCALL statement in the LCF.
udata	Call user data. If the destination host is a NOS system running network products, the first 12† octets must be of the form sss dd aaaaaaa, where: sss — is the 3 character ASCII equivalent of the SNODE (sendng node number) value, right-justified, zero filled. dd — is the 2 character ASCII equivalent of the DHOST (destination host number) value, right-justified, zero filled. aaaaaaa — is the 7 character ASCII equivalent of the called application's application name, left-justified, blank filled. The remainder of the udata field (0-112 octets) is passed to the called application as user data. The called host/application (if accessed through an X.25 network) must be able to support the Fast Select Facility, if more than 12 octets of information are specified. Note: For applications accessing foreign hosts through an X.25 network, the 4 octets of the PRID field and the (up to) 124 octets of the UDATA field are combined into the (up to) 128 octets of used data as defined by the CCITT recommendation for X.25 networks. You cannot access this field with NFETCH.

†An octet is 8 bits of information.

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

	59	51	49		43	35	17	0
ta	con	1	0	acrq	rc	abn	reserved	
	name1						name2	

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

con — Primary function code 63_{16}. You can access this field 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. You can access this field 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. The field is actually made up of two 4 bit subfields, rc1 and rc2. The rc1 field comprises bits 40 through 43 and the rc2 field comprises bits 36 through 39.

The rc2 field is used so that the application can determine what action to take when it receives a CON/ACRQ/A message and it provides some general information about the source of the trouble. This field can have the following values:

1 = Critical error in call request detected by source host (only LID/PID/NDL configuration changes or application code changes can solve the problem).

2 = Critical error in call request detected by destination host.

3 = Source host temporarily cannot make the connection (resources are currently not available, but they might become available without operator intervention).

4 = Destination host temporarily cannot make the connection.

5 = Source host cannot make the connection for an indefinite period of time (resources can be made available by operator intervention such as enabling a LID/PID, network element, or bringing up a system or subsystem).

6 = Destination host cannot make the connection for an indefinite period of time.

Thus if rc2 = 1 or 2, the application should not try establishing the connection again; it should notify the user and/or host operator that the connection is not possible.

If rc2 = 3 or 4, then the application can retry the CON/ACRQ message after a short time, and if rc2 = 5 or 6, then it can retry the CON/ACRQ after a longer time.

The rc1 field is used in combination with the rc2 field to uniquely identify the exact source of the trouble, so that the user/operator can take the appropriate action to fix the problem. The full 8-bit reason code field can therefore have the following values:

2 = Network error detected by destination host. Contact system analyst at destination host.

4 = Connection number conflict between source and destination host. Retry connection request.

17 = Invalid LID/PID combination was specified. Correct LID/PID in OUTCALL block.

18 = Called application is not defined in system record (CONTNAP) at destination host. Contact system analyst.

19 = Network Validation Facility (NVF) temporarily cannot process connection request. Retry later.

20 = Called application cannot accept any more connections and another copy of the application cannot be started up. Retry later.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format (Sheet 1 of 4)

Neither application program can send or receive any supervisory messages or data blocks 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 the message sequences shown in figure 3-15, a logical-error asynchronous supervisory message is issued. This message is discussed at the end of this section.

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; for example, the same logical connection can have 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).

Privileged applications can specify OUTCALL parameters in optional words 2-10 of the CON/ACRQ/R sequence. This allows the aplications to have more control over an outgoing call request. The application specifies a complete OUTCALL block except for the SNODE, DNODE, PORT, and DTE address parameters. NAM obtains these parameter values from the first OUTCALL statement defined in the LCF that has a matching NAME2 (PID).

Application	NAM	Message
←―――――――――――――		FC/INACT/R

The timer for the logical connection is reset to zero.

Application 1	NAM	Application 2	Message
―――――――――	――――――――→		FC/INACT/R

The timer for the logical connection is reset to zero.

Figure 3-15. Connection Monitoring Message Sequences

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 network 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 transmission on a logical connection, the network software uses one of the supervisory message sequences shown in figure 3-15 to inform the application program of the condition.

The connection monitoring sequence consists of the asynchronous inactive-connection message shown in figure 3-16. 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.

	59	51	49	43	35	23	0
ta	fc	0	0	inact	unused	acn	unused

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

fc
: Primary function code 83_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol FCACN, as described in section 4.

Figure 3-16. Inactive-Connection (FC/INACT/R) Supervisory Message Format

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 3-17.

Logical connection termination is initiated by the network whenever such conditions as hardware failure, a dialup line being disconnected without a formal logout by a terminal operator, and failure of another (connected) application program occur. 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.

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 network 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	NAM	Message
——————————→		CON/END/R
←——————————		CON/END/N

The logical connection is terminated by the application program. The application connection number can be reassigned to another logical connection by the network software.

Application	NAM	Message
←——————————		CON/CB/R

The logical connection is terminated by the network. The application program can salvage data in transit by fetching any blocks queued.

——————————→		CON/END/R
←——————————		CON/END/N

The application connection number can be reassigned to another logical connection by the network software.

Figure 3-17. Connection Termination Message Sequences

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 3-17. 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 batch or interactive console device.

When a zero aname parameter is used, a console 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, a console 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 that program, the console connection is not switched. In this case, a console is informed of the condition with the message APPLICATION NOT PRESENT or USER ACCESS NOT POSSIBLE - CONTACT NETWORK ADMIN. The terminal operator is then prompted for another application program name, unless the console was configured for a full automatic login procedure and the user name in that procedure validates for access only to the disconnected application program. In this instance, all of the terminal's ended logical connections are 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 device is a batch or interactive one. The HELLO or LOGIN command causes NVF to initiate a manual login dialog with an interactive device. The BYE or LOGOUT command causes NVF to disconnect a console device from the host.

When your program ends a connection with a passive device (a batch device of device types 1 through 4), any aname value you supply is ignored. NVF disposes of the passive device connection in the same manner as it does the device's owning console connection. That is:

If your program already disconnected the owning console for the device, NVF attempts to connect the device to the same program as the owning console; if the owning console is disconnected from the host, NVF disconnects the passive device as well.

If your program has not already disconnected the owning console for the device, NVF attempts to reconnect the device to your program. If your program rejects the reconnection, NVF keeps the device connected to itself until your program disconnects the owning console for the device.

On dialup lines, consoles without connections to hosts are assigned to a disconnection queue. When all consoles 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 devices on the line, including any passive devices 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 devices can be reconnected after a new dial-in procedure.

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

While a console is queued for disconnection, any terminal operator keyboard entry removes all the devices of that terminal from the disconnection queue and reconnects them 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 3-18, control list polling and list assignment. The other sequences, shown in figure 3-19, control the duplexing mode used during list processing.

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.

Application	NAM	Message
———————————→		LST/OFF/R

When the list number associated with the affected logical connection is next polled by the application program, no blocks will be returned from the connection.

Application	NAM	Message
———————————→		LST/ON/R

When the list number associated with the affected logical connection is next polled by the application program, blocks might be returned from the connection.

Application	NAM	Message
———————————→		LST/SWH/R

When the new list number associated with the affected logical connection is next polled by the application program, blocks might be returned from the connection.

Figure 3-18. Connection List Polling Control Message Sequences

Application	NAM	Message
———————————→		LST/FDX/R

When the list number associated with the affected logical connection is next polled by the application program, blocks can be returned from the affected logical connection regardless of the previous types of blocks output on the connection.

Application	NAM	Message
———————————→		LST/HDX/R

When the list number associated with the affected logical connection is next polled by the application program, blocks of application block type 1 or a single block of block type 2 are returned from the affected connection only if a block of block type 2 or a LST/ON/R message has been sent downline on the connection since the last upline block of block type 2 was delivered to the program. In effect, message input to the program is disabled until message output is complete.

Figure 3-19. Connection List Duplexing Message Sequences

If the program requires the list to be polled without returning any blocks queued from the connection, the asynchronous supervisory message shown in figure 3-20 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 3-21.

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 loop 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 in the polling process.

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 3-22. 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.

230 = Problem detected by X.25 network. X.25 network CCC=9. Destination host out of order. Wait until destination comes back up; then retry.

242 = Problem detected by X.25 network. X.25 network CCC=29. Fast select not subscribed to. Change OUTCALL portion of CON/ACRQ/R to not use fast select.

You can access this field with the reserved symbol RC, as described in section 4.

abn — Application block number. This field contains the abn of the previous CON/ACRQ/R message if there was one; otherwise, this field contains a zero. You can access this field with the reserved symbol CONAABN, as described in section 4.

reserved — Reserved by CDC.

name1 — Outcall identifier, 1 to 7 6-bit display code letters or digits (the first must be a letter), left-justified and blank-filled. This parameter is used to uniquely identify the appropriate OUTCALL definition that establishes a connection to another application. You can access this field with the reserved symbol CONANM, as described in section 4.

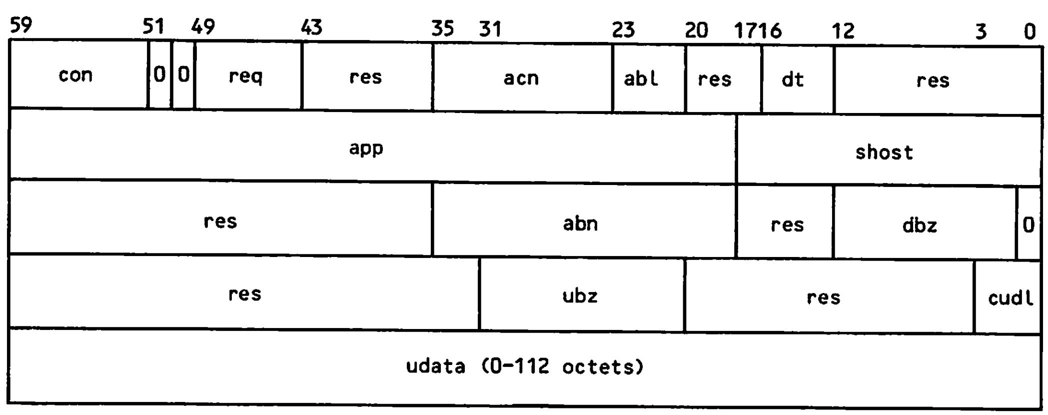
name2 — Outcall identifier, 1 to 3 display code letters or digits, left-justified and blank-filled. This parameter is optional (see the LID parameter). When explicitly specified in the CON/ACRQ message, or when implied by the LID together with NAME1; it is used to select the appropriate OUTCALL definition from the collection of outcall definitions previously specified by the Network Definition Language OUTCALL statement during the creation of the local configuration file (LCF). The combination of NAME1 and NAME2 (implicit or explicit) must appear as NAME1 and NAME2 or PID on an OUTCALL statement. For intra-host connections, both the LID and the PID can be zero.

If the application supplies its own outcall block, then the explicit or implicit PID must have appeared on a PID parameter in the OUTCALL statement of a previously created LCF.

You can access this field with the reserved symbol CONANM2, as described in section 4.

Figure 3-13. Application-Connection-Reject (CON/ACRQ/A) Supervisory Message Format (Sheet 4 of 4)

ta



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

con — Primary function code 63_{16}. You can access this field 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. You can access this field 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 contain zero.

Figure 3-14. Connection-Request (CON/REQ/R) Supervisory Message Format, Application-to-Application Connections (Sheet 1 of 2)

acn	Application connection number assigned to this logical connection; 1 ≤ minacn ≤ acn ≤ maxacn ≤ 4095, where minacn and maxacn are minimum and maximum values established by the application program in its NETON call. (See section 5.) You can access this field 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 when the connection is described in the local configuration file. If your application program initiated the connection request, this value comes from the ABL parameter of the NDL OUTCALL statement used by your program; if another application program initiated the connection request, the initial value comes from the ABL parameter of the NDL INCALL statement used by that program. This value is also supplied from the ab1 in the CON/ACRQ if the application supplies its own OUTCALL parameters. This field has the range 1 ≤ abl ≤ 7. You can access this field with the reserved symbol CONABL, as described in section 4.
dt	Device type of the connection. This field can have the values: 5 Application-to-application connection within the same host 6 Application-to-application connection between two hosts You can access this field with the reserved symbol CONDT, as described in section 4.
app	Application name. This field contains the application name of the other application program for intrahost application-to-application connections; otherwise, this field contains zero.
shost	Source host identifier. This field contains the node number of the host in which the other application program runs if this CON/REQ/R is received by the called application. The value is in 6-bit display code characters, left-justified with blank fill. The calling application receives a CON/REQ/R with the name2 field of the previous CON/ACRQ/R message or the name2 value of the corresponding OUTCALL parameter block.
abn	Application block number. This field contains the abn value assigned by your application program to the CON/ACRQ/R supervisory message if your program initiated the connection request; otherwise, this field contains a zero. You can access this field with the reserved symbol CONABN, as described in section 4.
dbz	Downline block size. The recommended maximum number of 8-bit character bytes in any network data block sent on the connection. If your application program initiated the connection request, this value comes from the DBZ parameter of the NDL OUTCALL statement used by your program; if another application program initiated the connection request, the initial value comes from the DBZ parameter of the NDL INCALL statement used by that program. This field can have the values 1 ≤ dbz ≤ 2043. You can access this field with the reserved symbol CONDBZ, as described in section 4.
ubz	Upline block size. The number of 8-bit bytes (in multiples of 100) the network will deliver in each upline network data block on the connection. If your application program initiated the connection request, this value comes from the UBZ parameter of the NDL OUTCALL statement used by your program. If another application program initiated the connection request, the initial value comes from the UBZ parameter of the NDL INCALL statement used by that program. This field can have the values 0 ≤ ubz ≤ 20, where 0 and 1 both indicate 100-byte blocks. If ubl is not specified, the default value of 2 is used. You can access this field with the reserved symbol CONUBZ, as described in section 4.
cudl	The call for the user's data length expressed in the number of octets. This field is set to zero if there is no call user data. You can access this field with the reserved symbol CONUDL, as described in section 4.
udata	Optional call user data. This is the call user data specified by the calling application in the CON/ACRQ/R supervisory message from a NOS host; or, it is the 13th through 128th octets of call user data an X.25 network. Allows applications to send a small amount of data to each other without actually establishing a connection via the fast select facility on an X.25 network. You cannot access this field with NFETCH.

Figure 3-14. Connection-Request (CON/REQ/R) Supervisory Message Format, Application-to-Application Connections (Sheet 2 of 2)

	59	51	49	43	35	23	0
ta	lst	0	0	fdx	unused	acn	unused

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

lst — Primary function code $C0_{16}$. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol LSTACN, as described in section 4.

Figure 3-24. Turn-On-Full-Duplex-List-Processing (LST/FDX/R) Supervisory Message Format

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.

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 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 a device or delivered to another application. Since NAM and CCP must save these extra blocks until they are processed by the other end of the connection, 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 the other end of the connection is underoccupied. Network software provides a set of conventions that allow the application to control the flow of data between itself and its connections 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 queue for delivery to the device or application, a block-delivered asynchronous supervisory message (figure 3-25) 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 error, 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 communication line, for example, fails before a block is completely output on a terminal device, the application program might still receive a block-delivered message.

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 3-26) 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.

	59	51	49	43	35	23	5	0
ta	fc	0	0	ack	unused	acn	abn	unused

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

fc Primary function code 83_{16}. You can access this field 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. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol FCABN, as described in section 4.

Figure 3-25. Block-Delivered (FC/ACK/R) Supervisory Message Format

	59	51	49	43	35	23	5	0
ta	fc	0	0	nak	rc	acn	abn	unused

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

fc Primary function code 83_{16}. You can access this field 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. You can access this field 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 values:

0 Reserved for CDC use.

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.

2 thru 255 Reserved for CDC use.

You can access this field 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. You can access this field 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. You can access this field with the reserved symbol FCABN, as described in section 4.

Figure 3-26. Block-Not-Delivered (FC/NAK/R) Supervisory Message Format

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.

When a break supervisory message is received for an application-to-application connection, set K(i)=0.

When a user-break caused user-interrupt supervisory message is received for a device-to-application connection, do not set K(i)=0; block-delivered messages make this unnecessary.

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

The break and user-break caused user-interrupt supervisory messages included in this strategy affect downline traffic on a logical connection. (The break message also affects upline traffic.) Such messages are sent to the application program whenever a network condition requires downline transmission on the connection to be interrupted.

The NPU relies on the application program to decide when traffic can be resumed. Two sequences of events are possible when such interruptions occur. The sequence that occurs depends on whether the connection involved is with another application program or with a terminal device.

For application-to-application connections, the following happens (see figure 3-27):

1. Blocks sent downline by your application program but not yet delivered to the other application are discarded.

2. Blocks sent upline to your application program but not yet delivered from the other application program are discarded.

3. An asynchronous break supervisory message (figure 3-28) is sent to your application program. If the connection uses an X.25 communication line, the side of the X.25 network originating the break is indicated by a reason code in the message.

4. Your application program resets its flow control algorithm, as described previously in this subsection.

5. Your application program issues an asynchronous reset supervisory message, as shown in figure 3-29, as a response to the break message. Until the reset message is sent, no upline or downline data can be exchanged on the connection. NAM sends no response to your reset message.

6. Normal downline (and upline) traffic can now resume. The first block sent or received on the connection that is not a null block marks the point in traffic where data flow was interrupted.

Application	NAM	Message
◄──────────	─────	FC/BRK/R
The network software discards all unacknowledged blocks queued for delivery to the other application.		
──────────	─────►	FC/RST/R
The application program can now resume communication with the other application.		

Figure 3-27. Application-to-Application Connection Break and Reset Message Sequence

For device-to-application connections, the following happens (see figure 3-30):

1. Blocks sent downline by your application program but not yet delivered to the device are discarded. Discarded blocks are acknowledged as delivered by NAM.

2. NAM sends an asynchronous user-interrupt supervisory message with a reason code indicating a user-caused break (figure 3-31) to your application program.

3. NAM queues a synchronous break-indication-marker supervisory message (figure 3-32) after any data blocks not yet delivered to your application program.

4. Your application program issues an asynchronous interrupt-response supervisory message, as shown in figure 3-33, as a response to the user-interrupt message. Until this response message is sent, additional user-interrupt conditions involving the device are ignored. NAM sends no response to your user-interrupt-response message.

5. Your application program processes all pending input on that connection by issuing NETGET or NETGETF calls (section 5) until the break-indication-marker message is received. The disposition of received data blocks is up to your application program.

6. Your application program issues a synchronous resume-output-marker supervisory message (figure 3-34), as a response to the break-indication-marker message. Until this message is sent, downline data sent on the connection is discarded by the network. NAM sends no response to your resume-output-marker message. Normal downline traffic can now resume.

If your application program does not complete one of these sequences properly, it receives an asynchronous logical-error supervisory message. The logical-error message is described at the end of section 3.

The user-interrupt message reflects suspension of downline traffic only. Upline traffic (input) on the connection is not affected.

	59	51	49	43	35	23	0
ta	fc	0	0 brk	rc	acn	reserved	

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

fc — Primary function code 83_{16}. You can access this field 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. You can access this field 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 is nonzero in upline messages for X.25 connections only. This field can contain the values:

1 thru 4	Reserved for CDC use.
5	A data communications equipment (DCE) break indicator (reset indication packet) occurred for the X.25 communication line used by the connection.
6	A data terminal equipment (DTE) break indicator (reset indication packet) occurred for the X.25 communication line used by the connection.
8 thru 191	Reserved for CDC use.
192 thru 255	Reserved for site-defined use.

You can access this field with the reserved symbol FCRBR, 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. You can access this field with the reserved symbol FCACN, as described in section 4.

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

Figure 3-28. Break (FC/BRK/R) Supervisory Message Format

	59	51	49	43	35	23	0
ta	fc	0	0 rst	reserved	acn	reserved	

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

fc — Primary function code 83_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol FCACN, as described in section 4.

Figure 3-29. Reset (FC/RST/R) Supervisory Message Format

Application	NAM	Message	Connection
◄────────		INTR/USR/R	Zero

The network software acknowledges and discards all blocks queued for delivery to the device. Your application program can request queued input from NAM but cannot receive another INTR/USR/R affecting this connection.

The program requests all queued input from NAM. The network software continues to discard and acknowledge downline blocks.

Application	NAM	Message	Connection
◄────────		BI/MARK/R	Nonzero
────────►		INTR/RSP/R	Zero
────────►		RO/MARK/R	Nonzero

Your application program can now resume output to the device. NAM stops discarding downline blocks.

Figure 3-30. Terminal User-Caused Break Sequence

	59	51	49		43	35	23
ta	intr	0	0	usr	rc	acn	unused (bit 0 at right)

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

intr — Primary function code 80_{16}. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol USR.

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

- 0 thru 2 — Valid on application-to-application connections only; no predefined meaning.
- 3 — On device-to-application connections, the terminal operator used the key or entered the character defined for the device as generating a user-break-1 condition; discard all blocks received until a BI/MARK/R synchronous supervisory message is received. On application-to-application connections, no predefined meaning.
- 4 — On device-to-application connections, the terminal operator entered the character defined for the device as generating a user-break-2 condition; discard all blocks received until a BI/MARK/R synchronous supervisory message is received. On application-to-application connections, no predefined meaning.
- 5 thru 255 — On device-to-application connections, refer to figure 3-39. On application-to-application connections, no predefined meaning.

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

Figure 3-31. User-Interrupt (INTR/USR/R) Supervisory Message Format

	59	51		49	43 — 0	
ta	bi	0	0	mark	unused	act=2

	59	55	47	43		41	35 — 0	
ta	0	bi	0	0	0	mark	unused	act=3

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

bi Primary function code CA_{16}. You can access this field with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol BI.

mark Secondary function code 0. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol MARK.

Figure 3-32. Break-Indication-Marker (BI/MARK/R) Supervisory Message Format

	59	51		49	43	35	23 — 0
ta	intr	0	0	rsp	0	acn	unused

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

intr Primary function code 80_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol INTRACN, as described in section 4.

Figure 3-33. Application-Interrupt-Response (INTR/RSP/R) Supervisory Message Format

	59	51		49	43 — 0	
ta	ro	0	0	mark	unused	act=2

	59	55	47	43		41	35 — 0	
ta	0	ro	0	0	0	mark	unused	act=3

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

ro Primary function code CB_{16}. You can access this field with the reserved symbol PFC, as described in section 4. Its value is defined as the value of the reserved symbol RO.

mark Secondary function code 0. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol MARK.

Figure 3-34. Resume-Output-Marker (RO/MARK/R) Supervisory Message Format

CONTROLLING OR BYPASSING UPLINE AND DOWNLINE DATA

Several asynchronous supervisory messages allow your application program to:

Control the flow of upline and downline data to both ends of an application-to-application connection.

Control the flow of downline data on a device-to-application connection.

Bypass data blocks or synchronous supervisory messages on an application-to-application connection; this allows your application program to control the flow of downline data on an application-to-application connection if both programs recognize a method of doing so.

The sequences and forms of the messages used depend on whether the connection is with another application program or with a terminal device.

Discarding Upline and Downline Data on Application-to-Application Connections

Your program can discard all upline and downline data queued between itself and another application program by sending the asynchronous break supervisory message shown in figure 3-28. NAM does not send a response for this message to your program.

The rest of the steps shown in figure 3-27 then occur:

1. Blocks sent downline by each application program but not yet delivered to the other application are discarded.

2. Blocks sent upline to each application program but not yet delivered from the other application program are discarded.

3. An asynchronous break supervisory message (figure 3-28) is sent to the other application program.

4. Each application program resets its flow control algorithm, as described previously under Monitoring Downline Data.

5. The other (receiving) application program issues an asynchronous reset supervisory message, as shown in figure 3-29. Until the reset message is sent, no upline or downline data can be exchanged on the connection. NAM sends no response to either reset message.

6. Normal downline and upline traffic can now resume. The first block sent or received on the connection that is not a null block marks the point in traffic where data flow was interrupted.

Discarding Downline Data on Device-to-Application Connections

Your program can discard all downline data queued between itself and a terminal device by sending the asynchronous application-interrupt supervisory message shown in figure 3-35, using a parm field value of 2.

The first set of steps shown in figure 3-36 then occurs:

1. The network begins discarding downline blocks queued for delivery to the device. Upline blocks queued for delivery to your application program are not affected.

2. Your application program sends a synchronous terminate-output-marker supervisory message, as described in figure 3-37. This message indicates to the network software the place in the downline data flow where it should stop discarding blocks.

3. The network sends your application program an asynchronous interrupt-response supervisory message (figure 3-33). Until this message is received, your program cannot send another application-interrupt message affecting the same connection.

4. Normal downline data traffic can now resume.

If your application program issues another application-interrupt message before receiving an interrupt-response message, it receives an asynchronous logical-error supervisory message. The logical-error message is described at the end of section 3.

Bypassing Downline Data on an Application-to-Application Connection

Your program can bypass all downline data queued between itself and another application by sending the asynchronous application-interrupt supervisory message shown in figure 3-37, using any parm field value. NAM does not send a response for this message to your program.

The second set of steps shown in figure 3-38 then occurs:

1. The network does not discard any blocks queued for delivery to the other application program. Upline blocks from the other program queued for delivery to your application program are not affected. Neither program's flow control algorithm is affected.

2. The network sends the other application program an asynchronous user-interrupt supervisory message (figure 3-31), containing a reason code equal to the parm value your program sent in its application-interrupt message.

3. The other application program sends the network an asynchronous interrupt-response supervisory message (figure 3-33). If the other program recognizes the reason code as indicating the need to discard your program's downline (the other program's upline) data blocks, it will begin to do so.

	59	51		49	43	35	23	0
ta	intr	0	0	app	parm	acn	0	

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

intr Primary function code 80_{16}. You can access this field 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. You can access this field 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. Can be one of the following:

0 and 1 Valid on application-to-application connections only; no predefined meaning.

2 On device-to-application connections, discard all blocks received until a TO/MARK/R synchronous supervisory message is received. On application-to-application connections, no predefined meaning.

3 thru 255 Valid on application-to-application connections only; no predefined meaning.

You can access this field with the reserved symbol INTRCHR, as described in section 4.

acn Application connection number assigned by the network software for the connection on which the application interrupt is requested. You can access this field with the reserved symbol INTRACN, as described in section 4.

Figure 3-35. Application-Interrupt (INTR/APP/R) Supervisory Message Format

	59	51		49	43	35	23	0
ta	intr	0	0	rsp	0	acn	unused	

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

intr Primary function code 80_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol INTRACN, as described in section 4.

Figure 3-36. Application-Interrupt-Response (INTR/RSP/R) Supervisory Message Format

	59	51	49	43	0	
ta	to	0	0	mark	unused	act=2

	59	55	47	43	41	35	0	
ta	0	to	0	0	0	mark	unused	act=3

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

to Primary function code $C4_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol MARK.

Figure 3-37. Terminate-Output-Marker (TO/MARK/R) Supervisory Message Format

Application		NAM	Message	Connection
	→		INTR/APP/R	Zero

The network acknowledges and discards all blocks queued for delivery to the device. Your application program can request queued input from NAM but cannot send another INTR/APP/R affecting this connection.

Application		NAM	Message	Connection
	→		TO/MARK/R	Nonzero
	←		INTR/RSP/R	Zero

Your application program can now resume output to the device. NAM stops discarding downline blocks.

Application 1		NAM		Application 2	Message	Connection
	→				INTR/APP/R	Zero
			→		INTR/USR/R	Zero

The other application program discards all blocks delivered to it, if that is an appropriate action for an interrupt.

Application 1		NAM		Application 2	Message	Connection
	→	→	→		marker	Nonzero

Your application program can now resume normal output. The other program stops discarding your downline blocks.

Application 1		NAM		Application 2	Message	Connection
			←		INTR/RSP/R	Zero
	←				INTR/RSP/R	Zero

Figure 3-38. Downline Data Flow Control Supervisory Message Sequences

If your program does not use the application-interrupt message as a method of discarding data, the following step does not apply:

4. Both programs now must recognize some marker in your program's downline data to indicate the point in the process where the other program should stop discarding blocks. The synchronous terminate-output-marker supervisory message, as described in figure 3-36, can be used. NAM sends no response to this message and does not interpret it.

5. The other application program issues an interrupt-response asynchronous supervisory message (figure 3-33).

6. The network sends your application program an asynchronous interrupt-response supervisory message (figure 3-33). Until this message is received, your program cannot send another application-interrupt message affecting the same connection.

7. Your program can now resume normal downline traffic.

TERMINAL USE OF USER INTERRUPTS FOR PRIORITY DATA

The terminal operator can send a message to the application that bypasses regular upline data by entering a user-interrupt priority data sequence. The operator enters the sequence by entering the TIP command control character (defined by the CT command) and an alphabetic character. NAM generates the user-interrupt-request supervisory message, INTR/USR/R (illustrated in figure 3-39) and sends it to the application.

The application program responds with the application-interrupt-response supervisory message (illustrated in figure 3-36) after receiving the INTR/USR/R message if the application supports user interrupts. If the application does not support priority data user interrupts, it can ignore the INTR/USR/R message and issues no response. Figure 3-40 illustrates the flow of messages. Until the response is sent, the user cannot enter another interrupt sequence.

Application	NAM	Message
◄──────────		INTR/USR/R

NAM delivers the user-interrupt ASCII character to the application in an asynchronous supervisory message on acn=0.

Supervisory programs and applications that do not support the user-interrupt-request message need take no further action.

Application	NAM	Message
──────────►		INTR/RSP/R

The application that supports user interrupt requests must respond with an interrupt-response supervisory message on acn=0.

Figure 3-40. User Interrupt for Priority Data Supervisory Message Sequence

If the application program supports priority data user interrupts, predefined meanings can be given to the ASCII characters available as interrupt characters. Only the characters A through Z and a through z can be used.

	59	51	49	43	35	23	0
ta	intr	0	0	usr	char	acn	unused

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

intr — Primary function code 80_{16}. You can access this field 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. You can access this field 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 character. This 8-bit field contains one of the 7-bit ASCII codes for letters shown in table A-2. You can access this field 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. You can access this field with the reserved symbol INTRACN, as described in section 4.

Figure 3-39. User-Interrupt-Request (INTR/USR/R) Supervisory Message Format for Priority Data

CONTROLLING UPLINE BLOCK CONTENT

Several asynchronous supervisory messages allow you to control the content of upline blocks. (Downline block content is controlled directly by your program and indirectly by the values your program places in the accompanying application block header.) Using supervisory messages, your program can:

- Convert character codes in unreceived upline network data blocks to 6-bit display code or cancel such conversion
- Change character byte packing in unreceived upline network data blocks
- Change byte packing in unreceived synchronous supervisory message blocks
- Discard unreceived transparent mode data from a device or cancel that discarding operation
- Truncate unreceived upline blocks

The following subsections describe these supervisory messages.

CONVERTING AND REPACKING DATA

Data exchanged on an interactive device-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 device 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.

Upline data 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 the connection 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 3-41, 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 3-41. Change-Input-Character-Type Supervisory Message Sequence

The change-input-character-type message has the format shown in figure 3-42. 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.

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.

	59	51	49		43	35	23	7		5 0
ta	dc	0	0	cict	unused	acn	unused	nxp	sct	act

ta	Symbolic address of the application program's text area from which this asynchronous supervisory message is sent.
dc	Primary function code $C2_{16}$. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol DCACN, as described in section 4.
nxp	No-transparent-input flag. This field can have the values: 0 Deliver network data blocks with the xpt bit set in the associated block header 1 Do not deliver network data blocks with the xpt bit set in the associated block header You can access this field 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 application character type 2 1 Deliver supervisory messages in application character type 3 You can access this field with the reserved symbol DCSCT, as described in section 4.

Figure 3-42. Change-Input-Character-Type (DC/CICT/R) Supervisory Message Format (Sheet 1 of 2)

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:

0 or 1	Reserved for CDC use.
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 use.
12 thru 15	Reserved for installation use.

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. You can access this field with the reserved symbol CONACT, as described in section 4.

Figure 3-42. Change-Input-Character-Type (DC/CICT/R) Supervisory Message Format (Sheet 2 of 2)

REPACKING SYNCHRONOUS SUPERVISORY MESSAGE BLOCKS

Synchronous supervisory message block fields are packed in either 8-bit or 12-bit bytes, at the discretion of the application program. NAM packs or unpacks fields in a given synchronous supervisory message block according to the application character type associated with the block (downline) or with the connection (upline).

Synchronous supervisory messages sent downline by an application program have an application character type associated with them on a block-by-block basis. When the application program needs to change the packing performed for blocks on a given connection, it simply changes the act field value used in the block header of each synchronous supervisory message. The effects of a given act field value declaration are described in detail in section 2.

An upline synchronous supervisory message block has an application character type associated with the connection on which the block is received. The application character type associated with the connection is assigned by the application program as the sct field value when the connection is first established. This assignment is part of the connection-accepted supervisory message and is separate from the assignment made for data blocks received on the connection.

When the application program needs to change the packing performed for upline synchronous supervisory messages on a given connection, it changes the sct field value associated with the 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 3-41, there is no response to the change-input-character-type supervisory message, but the message takes effect immediately.

The change-input-character-type message has the format shown in figure 3-42. The application character types selected with the sct field values are described in more detail in section 2.

The repacking 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 synchronous supervisory messages with fields packed in 12-bit bytes (using an application character type of 3); the application program now needs to receive synchronous supervisory message blocks with fields stored in 8-bit bytes (using an application character type of 2). The program sends a change-input-character-type message, specifying an sct field value of 0; the next synchronous supervisory message block received on that logical connection is packed in 8-bit bytes.

EXCHANGING TRANSPARENT DATA WITH DEVICES

Transparent data is exchanged with a terminal device at the discretion of the application program. NAM transfers transparent data blocks according to the transparent data flag associated with the block.

Network data blocks sent downline by an application program have a transparent data flag associated with them on a block-by-block basis. When the application program needs to change from or to transparent mode output on a given connection, it simply changes the xpt field value used in the application block header of each downline data block. The effects of a given xpt field value are described in detail in section 2.

Upline network data blocks also have a transparent data flag associated with them on a block-by-block basis. Each connection has a no-transparent-data flag associated with that connection. This flag indicates whether the application wants to receive transparent data or wants NAM to discard such data. The no transparent-data flag setting associated with the connection is assigned by the application program as the nxp field value when the connection is first established. This assignment is part of the connection-accepted supervisory message.

When the application program needs to change the value of the no-transparent-data flag for a given connection, it issues the change-input-character-type synchronous 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 3-41, there is no response to the change-input-character-type message, but the message takes effect immediately.

The change-input-character-type message has the format shown in figure 3-42. The effects of the nxp field values used in the message are described in section 2, where the application block header fields are described.

The transparent data exchange change requested by the change-input-character-type message affects the next upline block and all subsequent blocks queued for the application program. For example, the application program might have been receiving transparent blocks for an interactive console when the program contains no code to process those blocks; it needs to prevent receipt of any more transparent blocks while that connection exists. The program sends a change-input-character-type message, specifying an nxp field value of 1; the next (and any subsequent) block from that terminal device is discarded if it is in transparent mode, even if that block completes the current message.

The setting of the no-transparent-input flag does not cause data blocks on application-to-application connections to be discarded, unless the sending application program sets the xpt field value of the associated block header to 1.

TRUNCATING UPLINE BLOCKS

Blocks received upline by an application program from a terminal or from another application can be truncated to fit the text area buffer provided by your 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 can be used 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 3-43, the effect of this supervisory message cannot be reversed, and there is no response.

Application	NAM	Message
────────────►		DC/TRU/R

The next upline block delivered for this logical connection or all connections (depending on whether a nonzero acn is specified in the DC/TRU/R) will be truncated if necessary.

Figure 3-43. Block Truncation Supervisory Message Sequence

When a block is 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 block truncation supervisory message (figure 3-44) 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.

	59	51	49		43	35	23	0
ta	dc	0	0	tru	unused	acn	unused	

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

dc Primary function code $C2_{16}$. You can access this field 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}. You can access this field 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. You can access this field with the reserved symbol DCACN, as described in section 4.

Figure 3-44. Block Truncation (DC/TRU/R) Supervisory Message Format

MANAGING DEVICE CHARACTERISTICS

Devices serviced as interactive virtual terminals have many characteristics that can affect the way in which they send or output data. The network software can use varying numbers of these characteristics, depending on the terminal class of the device and sometimes on the protocol used by the device.

The following characteristics can be known and used through the network software when servicing an asynchronous device in terminal classes 1 through 8, or any device in terminal classes 28 through 31:

Character used to discard a block of output

Whether the break key should be interpreted as a cancel input and user break 1 command (does not apply to terminal class 4)

Backspace character used to edit a line of data

Characters used as user break 1 and user break 2 commands

Number of idle characters needed after a carriage return or a line feed

Character used to cancel an input line

Cursor positioning needed at the end of a physcial line or block (does not apply to terminal class 4)

Network control character used

Delimiters of single-message transparent input (does not apply to terminal class 4)

Delimiters of multiple-message transparent input (does not apply to terminal class 4)

Character used at the end of a logical input line or of an input block (does not apply to terminal class 4)

Echoplex mode (does not apply to terminal class 4)

Whether full-ASCII or special editing mode is in use

Whether the host availability display appears in full form

Whether the device supports input or output flow control characters (does not apply to terminal class 4)

Whether the device is using paper tape, a keyboard, block mode, or transparent mode during input (does not apply to terminal class 4)

Whether the device is using a display, a printer, or paper tape during output (paper tape does not apply to terminal class 4)

The parity processing required during input and output (does not apply to terminal class 4)

What the page width and page length are

Whether page waiting occurs

Whether unsolicited messages from the network operator can be delivered

What the terminal class is

Whether the communication line is serviced in full-duplex mode (does not apply to terminal class 4)

What the upline blocking factor is

What the transmission block size is

The following characteristics can be known and used through the network software when servicing an X.25 device in terminal classes 1 through 3 or 5 through 8:

Whether the break key should be interpreted as a user break 1 command

Backspace character used to edit a line of data

Characters used as user break 1 and user break 2 commands

Number of idle characters needed after a carriage return or a line feed

Character used to cancel an input line

Cursor positioning needed at the end of a physical line or block

Network control character used

Delimiters of single-message transparent input

Delimiters of multiple-message transparent input

Character used at the end of a logical input line or of an input block

Whether full-ASCII mode is in use

Whether the host availability display appears in full form

Whether the device is using a display, a printer, or paper tape during output

The parity processing required during output

What the page width and page length are

Whether page waiting occurs

Whether unsolicited messages from the network operator can be delivered

What the terminal class is

Whether the communication line is serviced in full-duplex mode (does not apply to terminal class 4)

What the upline blocking factor is

What the transmission block size is

The following characteristics can be known and used through the network software when servicing a CDC mode 4 device in terminal classes 10 through 13 or 15:

Characters used as user break 1 and user break 2 commands

Character used to cancel an input line

Network control character used

Delimiters of single-message transparent input

Delimiters of multiple-message transparent input

Character used at the end of a logical input line or of an input block

Whether full-ASCII editing mode is in use

Whether the host availability display appears in full form

Whether the device is using block mode or transparent mode during input

What the page width and page length are

Whether page waiting occurs

Whether unsolicited messages from the network operator can be delivered

What the terminal class is

What the upline blocking factor is

What the terminal transmission block size is

The following characteristics can be known and used through the network software when servicing a HASP device in terminal classes 9 or 14:

Characters used as user break 1 and user break 2 commands

Character used to cancel an input line

Network control character used

Character used at the end of a logical input line

Whether the host availability display appears in full form

What the page width and page length are

Whether page waiting occurs

Whether unsolicited messages from the network operator can be delivered

What the terminal class is

What the upline blocking factor is

What the terminal transmission block size is

The following characteristics can be known and used by the network software when servicing a 2780 or 3780 device in terminal classes 16 or 17:

Network control character used

What the page width and page length are

Whether page waiting occurs

Whether unsolicited messages from the network operator can be delivered

What the terminal class is

What the upline blocking factor is

What the terminal transmission block size is

The following characteristics can be known and used through the network software when servicing a 3270 device in terminal class 18:

- Characters used as user break 1 and user break 2 commands
- Character used to cancel an input line
- Network control character used
- Character used at the end of a logical input line
- Whether the host availability display appears in full form
- What the page width and page length are
- Whether page waiting occurs
- Whether unsolicited messages from the network operator can be delivered
- What the terminal class is
- What the upline blocking factor is
- What the terminal transmission block size is

Your application program can determine these characteristics or change them by using the supervisory messages described in the next subsections. Information on the use of these characteristics appears in the NAM 1/CCP 3 Terminal Interfaces reference manual listed in the preface.

CHANGING DEVICE CHARACTERISTICS

The process of configuring a terminal consists of defining a number of device characteristics that the network software should use in communication with a terminal. Some device 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 device is configured (or defined), subsequent changes to the device definition can be made via terminal definition commands from the terminal operator, or via supervisory messages from the application program to which the device is connected.

This subsection describes the supervisory messages that the application can use to change the settings of device characteristics. The supervisory message used to find out the current values of device characteristics is described in the following subsection, Requesting Device Characteristics. Terminal definition commands are described in the NAM 1/CCP 3 Terminal Interfaces reference manual listed in the preface.

Figure 3-45 shows the most probable message sequences involved in changing terminal characteristics.

The application program is advised of the terminal definition command entry explicitly only when the command changes one of three device characteristics:

> Terminal class (value describing the physical attributes of a group of similar terminals)
>
> Page width (value describing the number of characters in each physical line of output)
>
> Page length (value describing the number of physical lines output per page)

The upline terminal-characteristics-redefined supervisory message is an asynchronous one, with the format shown in figure 3-46. This message is sent to the application by NAM whenever NAM is notified that one of the three device characteristics has been redefined by a terminal user or by the application program. The effect of the terminal definition command causing this message is immediate, and no response is required from the application program.

Application	NAM	Message

The terminal operator enters the TC, PW, or PL commands to the Terminal Interface Program.

Application	NAM	Message
←———————		TCH/TCHAR/R

The next block sent to the device or from the device is affected by any constraints imposed under the new device page width, page length, or terminal class.

Application	NAM	TIP	Message

The application program changes a device characteristic other than page width, page length, or terminal class.

Application	NAM	TIP	Message
——————————————→			CTRL/DEF/R

The next block sent to the device or sent from the device is affected by any constraints imposed under the new device characteristic.

Application	NAM	TIP	Message

The application program changes page width, page length, or terminal class.

Application	NAM	TIP	Message
——————————————→			CTRL/DEF/R
←———————			TCH/TCHAR/R

The next block sent to the device or sent from the device is affected by any constraints imposed under the new page width, page length, or terminal class.

Figure 3-45. Terminal Characteristics Redefinition Supervisory Message Sequences (Sheet 1 of 2)

Application	NAM	Message

The application sends a define-multiple-terminal-characteristics message to NAM in order to redefine several of the terminal characteristics with a single message. The message is properly formatted and the new characteristics take effect immediately. NAM replies with a define-terminal-characteristics normal response.

Application → NAM	CTRL/CHAR/R
NAM → Application	CTRL/CHAR/N

Application	NAM	Message

The application sends a define-terminal-characteristics message to NAM, but one of the FN/FV pairs is bad. The changes do not take effect, and a define-terminal-characteristics abnormal response is sent to the application.

Application → NAM	CTRL/CHAR/R
NAM → Application	CTRL/CHAR/A

Figure 3-45. Terminal Characteristics Redefinition Supervisory Message Sequences (Sheet 2 of 2)

	59	51	49	43	35	23	15	7	0
ta	tch	0	0	tchar	unused	acn	tclass	pw	pl

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

tch — Primary function code 64_{16}. You can access this field 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. You can access this field 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. You can access this field with the reserved symbol CONACN, as described in section 4.

tclass — The terminal class currently associated with the real device by the TIP servicing it. 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 (see text). The tclass field can contain the values:

- 0 Reserved for CDC use.
- 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, 752, 756.
- 3 Archetype terminal for the class is a CDC 721.
- 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 VT100 (ANSI X3.64).

Figure 3-46. Terminal-Characteristics-Redefined (TCH/TCHAR/R) Supervisory Message Format (Sheet 1 of 2)

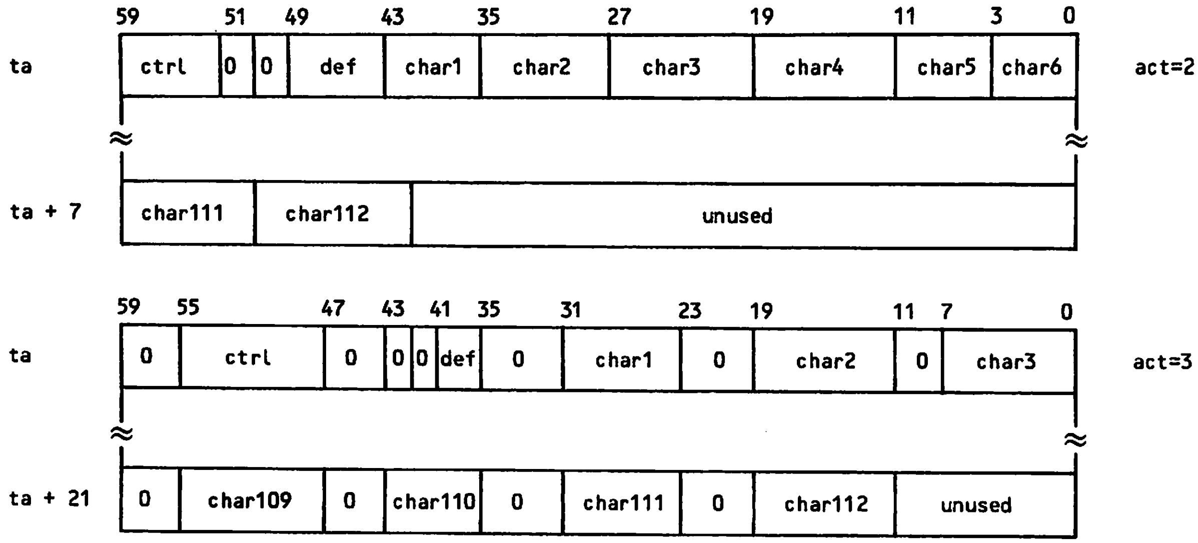
	8	Archetype terminal for the class is a Tektronix 4000 Series, operating as a teletypewriter.
	9	Archetype terminal for the class is a HASP (post-print) protocol multileaving workstation.
	10	Archetype terminal for the class is a CDC 200 User Terminal.
	11	Archetype terminal for the class is a CDC 714-30.
	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 work-station.
	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.
	18	Archetype terminal for the class is an IBM 3270.
	19 thru 27	Reserved for CDC use.
	28 thru 31	Site-defined terminal class.
	If the terminal class value received has not changed from that previously associated with the device, 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 device, then all attributes associated with the device 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. You can access this field with the reserved symbol TCHTCL, as described in section 4.	
pw	The most recently declared page width of the console device, specifying the number of characters in a physical line of output. This field can contain the values 0 or 20 $\leq$ pw $\leq$ 255. You can access this field with the reserved symbol TCHPW, as described in section 4.	
pl	The most recently declared page length of the console device, specifying the number of physical lines that constitute a page. This field can contain the values 0 or 8 $\leq$ pl $\leq$ 255. You can access this field with the reserved symbol TCHPL, as described in section 4.	

Figure 3-46. Terminal-Characteristics-Redefined (TCH/TCHAR/R) Supervisory Message Format (Sheet 2 of 2)

There are two different formats for changing terminal characteristics. Regardless of the format used, terminal class should only be changed before other changes are made. A change in terminal class resets many other characteristics.

The define-terminal-characteristics supervisory message (figure 3-47) 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.

The define-multiple-terminal-characteristics message is described in figure 3-48. 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 corresponding to a terminal definition command or command parameter, 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



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

ctrl — Primary function code $C1_{16}$. You can access this field with the reserved symbol PFC, as described in section 4. Its value is defined as the reserved symbol CTRL.

def — Secondary function code 4. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol DEFF.

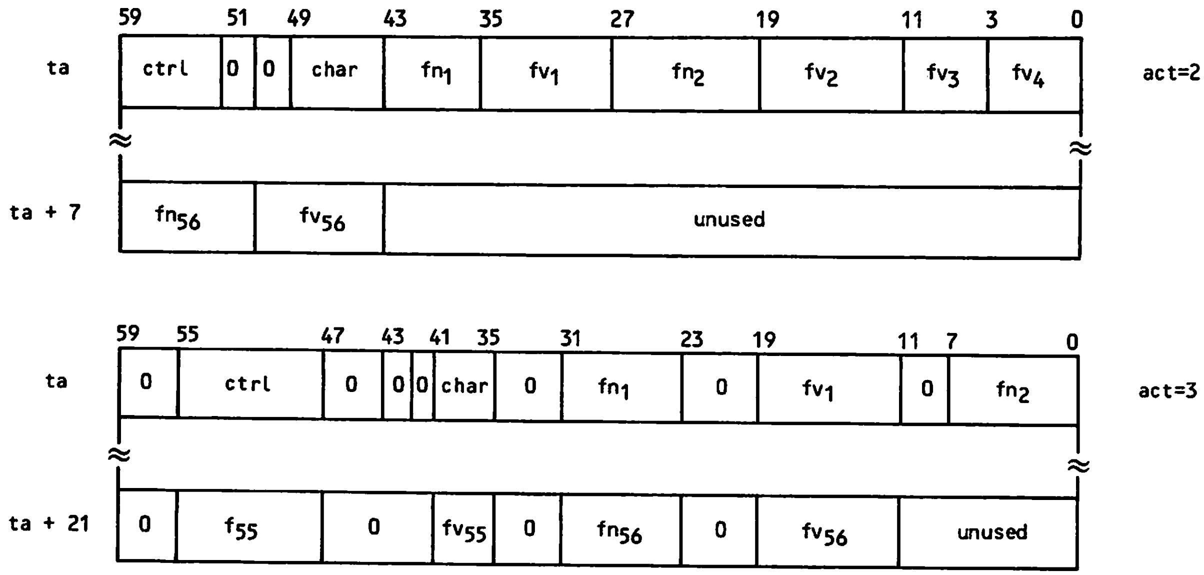
$char_i$ — Up to 112 7-bit ASCII characters of one or more commands consisting of the network control character, characteristic mnemonic, and its desired setting. The characteristic and its value are separated by an equals sign. Multiple characteristics can be changed by separating the commands with the network control character. See the Terminal Interfaces reference manual for the possible commands that can be sent.

Figure 3-47. Define-Terminal-Characteristics (CTRL/DEF/R) Supervisory Message Format

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 3-49. Figure 3-50 shows the normal response to the define-multiple-terminal-characteristics supervisory message.

Valid combinations of FN/FV pairs are defined in table 3-2. Field numbers are listed in hexadecimal, with octal equivalents in parentheses. Field values are listed only in hexadecimal.

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 $C1_{16}$. You can access this field 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. You can access this field 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 parameter to be changed.
fv_i	The 8-bit field value for the parameter.
	Up to 56 field number and field value pairs can be specified in a single message. Valid field numbers and values are defined in table 3-2.

Figure 3-48. Define-Multiple-Terminal-Characteristics (CTRL/CHAR/R) Supervisory Message Format

	59	51		49	43	35	27	0	
ta	ctrl	1	0	char	fn	rc	unused		act=3

	59	55	47	43	41		35	31	23	19	11	0	
ta	0	ctrl	0	1	0	char	0	fn	0	rc	unused		act=3

ta — Symbolic address of the application program text area receiving this synchronous supervisory message.

ctrl — Primary function code $C1_{16}$. You can access this field 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. You can access this field 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:

0	Reserved for CDC use.
1	Out of range value for command or parameter
2	Duplicate character definition
3	Invalid command or parameter value for terminal class to which device belongs
4	Illegal terminal class change
5	Illegal command or parameter for terminal class to which device belongs
6 thru 255	Reserved for CDC use

Figure 3-49. Define-Multiple-Terminal-Characteristics Abnormal Response (CTRL/CHAR/A) Supervisory Message Format

	59	51		49	43	0	
ta	ctrl	0	1	char	unused		act=2

	59	55	47	43	41		35	0	
ta	0	ctrl	0	0	1	char	unused		act=3

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

ctrl — Primary function code $C1_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is defined as the value of the reserved symbol CHAR.

Figure 3-50. Multiple-Terminal-Characteristics-Defined (CTRL/CHAR/N) Supervisory Message Format

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes (1)	Field Value Range	Field Value Content Meaning
Abort block (AB)	29 (51)	1 thru 8, (2) 28 thru 31 (9 thru 18)	0 thru 7E (3)	Numerical value for character
Blocking factor (BF)	31 (61)	1 thru 8, 10 thru 13, 15, 18 (1) (9, 14, 16, 17)	0 thru 20	Multiple of 100 characters that constitute an upline block
Break as user break 1 (BR)	33 (63)	1 thru 3, 5 thru 8, 28 thru 31 (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Backspace character (BS)	27 (47)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 7E (3)	Numerical value for character
User break 1 character (B1)	2A (52)	1 thru 15, 18, 28 thru 31 (16, 17)	0 thru 7E (3)	Numerical value for character
User-break-2 character (B2)	2B (53)	1 thru 15, 18, 28 thru 31 (16, 17)	0 thru 7E (3)	Numerical value for character
Carriage return idle count (CI)	2C (54)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 63	Number to insert
	2E (56)	1 thru 8, 28 thru 31 (9 thru 18)	1	TIP should calculate number
Cancel character (CN)	26 (46)	1 thru 15, 18, 28 thru 31 (16, 17)	0 thru 7E (3)	Numerical value for character
Cursor positioning (CP)	47 (107)	1 thru 3, 5 thru 8, 28 thru 31 (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Network control character (CT)	28 (50)	1 thru 18, 28 thru 31	0 thru 7E (3)	Numerical value for character
Single message (4) transparent input delimiters (DL)	38 (70)	1 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Character specified (1), not specified (0)
Message and mode delimiter	39 (71)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru 0F	Character count (upper byte)
Message and mode delimiter	3A (72)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru FF	Character count (lower byte)
Message and mode delimiter	3B (73)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16, 17)	0 thru FF (5)	Numerical value for character

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES (Contd)

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes (1)	Field Value Range	Field Value Content Meaning
Message and mode delimiter	3C (74)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Timeout (1), no timeout (0)
Mode type	46 (106)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31	0	Single message (0)
End-of-block character (EB)	40 (100)	1 thru 3, 5 thru 8, 10 thru 13, 15, 18, 28 thru 31	0 thru FF (5)	Numerical value for character
Use default terminator	41 (101)	1 thru 3, 5 thru 8, 10 thru 13, 15, 18, 28 thru 31	1 or 2 (5)	End-of-line (1), end-of-block (2)
End-of-block cursor positioning response	42 (102)	1 thru 3, 5 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16, 17, 18)	0 thru 3 (5)	No (0), CR (1), LF (2), CR and LF (3)
End-of-line character (EL)	3D (75)	1 thru 3, 5 thru 8, 10 thru 13, 15, 18, 28 thru 31	0 thru 7F (5)	Numerical value for character
Use default terminator	3E (76)	1 thru 3, 5 thru 8, 10 thru 13, 15, 18, 28 thru 31	1 or 2	End-of-line (1), end-of-block (2)
End-of-line cursor positioning response	3F (77)	1 thru 3, 5 thru 8, 10 thru 13 15, 28 thru 31 (9, 14, 16, 17, 18)	0 thru 3 (5)	No (0), CR (1), LF (2), CR and LF (3)
Echoplex mode (EP)	31 (61)	1 thru 3, 5 thru 8, 28 thru 31 (3) (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Full ASCII input (FA)	37 (67)	1 thru 8, 10 thru 13, 15, 16, 17, 18, 28 thru 31	0 or 1	Yes (1), no (0)
See host availability display (HD)	21 (41)	1 thru 18, 28 thru 31	0 or 1	Yes (1), no (0)
Input control (IC)	43 (103)	1 thru 3, 5 thru 8, 28 thru 31 (3) (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Input device (IN)	34 (64)	1 thru 8, 10 thru 13, 15, 28 thru 31	0 or 1	Transparent input (1), not transparent (0)
	35 (65)	1 thru 8, 28 thru 31 (6)	0 thru 2 (5)	Keyboard (0), paper tape (1), block mode (2)

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES (Contd)

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes (1)	Field Value Range	Field Value Content Meaning
Line feed idle count (LI)	2D (55)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 63	Number to insert
	2F (57)	1 thru 8, 28 thru 31 (9 thru 18)	1	TIP should calculate number
Lockout unsolicited messages (LK)	20 (40)	1 thru 15, 18, 28 thru 31 (16)	0 or 1	Yes (1), no (0)
Output control (OC)	44 (104)	1 thru 3, 5 thru 8, 28 thru 31 (2) (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Output device (OP)	36 (66)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 2 (5)	Display (0), printer (1), paper tape (2)
Parity processing (PA)	32 (62)	1 thru 3, 5 thru 8, 28 thru 31	0 thru 4	Zero (0), odd (1), even (2), none (3), ignore (4)
Page waiting (PG)	25 (45)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16, 17)	0 or 1	Yes (1), no (0)
Page length (PL)	24 (44)	1 thru 18, 28 thru 31	0, 8 thru FF (5)	Number of physical lines
Page width (PW)	23 (43)	1 thru 18, 28 thru 31	0, 20 thru FF	Number of characters
Site-defined use	90 thru 99 (220 thru 231)	1 thru 18, 28 thru 31	0 thru FF (5)	Site-defined
Special editing mode (SE)	30 (60)	1 thru 8, (6) 28 thru 31 (9 thru 18)	0 or 1	Yes (1), no (0)
Terminal class (TC)	22 (42)	1 thru 10, 28 thru 31	01 thru 0F (5)	Number of new class
Multiple-message (4) transparent delimiters (XL)	38 (70)	1 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Character specified (1), not specified (0)
Message delimiter	39 (71)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru F	Character count (upper byte)
Message delimiter	3A (72)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru FF	Character count (lower byte)
Message delimiter	3B (73)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16)	0 thru FF (5)	Numerical value for character

TABLE 3-2. VALID FIELD NUMBERS AND FIELD VALUES (Contd)

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes ①	Field Value Range	Field Value Content Meaning
Mode delimiter	3C (74)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Timeout (1), no timeout (0)
Mode delimiter	45 (105)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru FF ⑤	Numerical value for character
Mode type	46 (106)	1 thru 8, 10, 13, 15, 28 thru 31	1	Multiple-message (1)
Full duplex (none)	57 (127)	1 thru 3, 5 thru 8, 28 thru 31 (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Terminal transmission block size (none)	1E (36)	1 thru 18, ⑥ 28 thru 31	0 thru 7	Number of characters (upper byte)
	1F (37)	1 thru 18, ⑥ 28 thru 31	0 thru FF	Number of characters (lower byte)
Upline block limit (none)	18 (30)	1 thru 18, 28 thru 31	0 thru 1F ⑤	Number of blocks NPU should queue

Notes:

① No error occurs if an FN/FV pair is issued for a terminal class shown in parentheses.

② Ignored for CDC-defined X.25 packet assembly/disassembly (PAD) terminals.

③ Any hexadecimal value except 00 thru 02, 20, 30 thru 39, 3D, 41 thru 5A, 61 thru 7A, or 7F.

④ If the value of one of the fields for this command is changed, the values of all other fields for this command must also be specified.

⑤ Not all values are legal for all terminal classes.

⑥ Not allowed for CDC-defined X.25 packet assembly/disassembly (PAD) terminals.

REQUESTING DEVICE CHARACTERISTICS

The request-terminal-characteristics supervisory message (figure 3-51) is issued by an application program on console or site-defined device connections to learn the current value of the device 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 device characteristics are returned and the application receives the abnormal response message shown in figure 3-52. 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 3-53). This message can be received only on console or site-defined device 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.

59 51 49 43 35 27 19 11 0

ta	ctrl	0	0	rtc	fn_1	fv_1	fn_2	fv_2	...

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

ctrl — Primary function code $C1_{16}$. You can access this field 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. You can access this field 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 hexadecimal field number of the desired parameter. Valid values are defined in table 3-2.

fv_i — Space for the hexadecimal field value of the desired parameter; can be 0.

Figure 3-51. Request-Terminal-Characteristics (CTRL/RTC/R) Supervisory Message Format

59 51 49 43 35 27 0

ta	ctrl	1	0	rtc	fn	rc	unused

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

ctrl — Primary function code $C1_{16}$. You can access this field 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. You can access this field 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:

0 thru 4	Reserved for CDC use
5	Illegal field number value
6 thru 255	Reserved for CDC use

Figure 3-52. Request-Terminal-Characteristics Abnormal Response (CTRL/RTC/A) Supervisory Message Format

	59	51	49	43	35	27	19	11	0
ta	ctrl	0	0	tcd	fn_1	fv_1	fn_2	fv_2	...

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

ctrl — Primary function code $C1_{16}$. You can access this field 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 $0A_{16}$. You can access this field 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 hexadecimal field number of the characteristic parameter. Valid values are defined in table 3-2.

fv_i — The hexadecimal field value of the characteristic parameter. Valid values are defined in table 3-2.

Figure 3-53. Device-Characteristics-Definition (CTRL/TCD/R) Supervisory Message Format

HOST OPERATOR COMMANDS

The host operator can send commands to an application program through the system console K display. There are seven commands an application program might receive. Each command is delivered to the application program as a separate asynchronous supervisory message, as shown in figure 3-54.

The host operator request-to-activate-debug-code supervisory message (figure 3-55) is sent from NAM to the application program when the operator enters the K-display command:

K.DB=appname

The application should begin using any in-line debug code you have included. Activating 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 3-56 is sent from NAM to the application program when the operator enters the K-display command:

K.DE=appname

The application should turn off any in-line debug code you have included. There is no response to the request-to-turn-off-debug-code message.

The host operator request-to-dump-field-length supervisory message (figure 3-57) is sent from NAM to the application program when the operator enters the K-display command:

K.DU=appname

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-traffic-logging-on supervisory message (figure 3-58) is sent from NAM to the application program when the operator enters the K-display command:

K.LB=appname

The application program should call NETDBG to turn AIP logging on and begin logging of network traffic on the debug log file. (See section 6.) Note that the application program must be loaded with NETIOD for the AIP logging to occur. There is no response to the request-to-turn-AIP-traffic-logging-on message.

The host operator request-to-turn-AIP-traffic-logging-off supervisory message (figure 3-59) is sent from NAM to the application program when the operator enters the K-display command:

K.LE=appname

The application program should call NETDBG to turn AIP logging off and stop logging network traffic in its debug log file. (See section 6.) There is no response to the request-to-turn-AIP-traffic-logging-off supervisory message.

The host operator request-to-release-debug-log-file supervisory message (figure 3-60) is sent from NAM to the application program when the operator enters the K-display command:

K.LR=appname

Application	NAM	Message
←————————		HOP/DB/R

The program should begin using any debug code it contains.

Application	NAM	Message
←————————		HOP/DE/R

The program can stop using any debug code it contains.

Application	NAM	Message
←————————		HOP/DU/R

The program should dump its field length and any extended central storage.

Application	NAM	Message
←————————		HOP/TRACE/R

The program should begin using its debug log file.

Application	NAM	Message
←————————		HOP/NOTR/R

The program can stop using its debug log file.

Application	NAM	Message
←————————		HOP/REL/R

This program should release its debug log file for postprocessing.

Application	NAM	Message
←————————		HOP/RS/R

The program should reinitialize and restart logging of all of its statistics.

Figure 3-54. Host Operator Command Supervisory Message Sequences

	59	51	49		43 … 0
ta	hop	0	0	db	unused

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

hop Primary function code $D0_{16}$. You can access this field with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

db Secondary function code $0E_{16}$. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DB.

Figure 3-55. Host Operator Request-to-Activate-Debug-Code (HOP/DB/R) Supervisory Message Format

	59	51		49	43 … 0
ta	hop	0	0	de	unused

ta	Symbolic address of the application program's text area receiving this asynchronous supervisory message.
hop	Primary function code $D0_{16}$. You can access this field with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.
de	Secondary function $0F_{16}$. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DE.

Figure 3-56. Host Operator Request-to-Turn-Off-Debug-Code (HOP/DE/R) Supervisory Message Format

	59	51		49	43 … 0
ta	hop	0	0	du	unused

ta	Symbolic address of the application program's text area receiving this asynchronous supervisory message.
hop	Primary function code $D0_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol DU.

Figure 3-57. Host Operator Request-to-Dump-Field-Length (HOP/DU/R) Supervisory Message Format

	59	51		49	43 … 0
ta	hop	0	0	trace	unused

ta	Symbolic address of the application program's text area receiving this asynchronous supervisory message.
hop	Primary function code $D0_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol TRACE.

Figure 3-58. Host Operator Request-to-Turn-AIP-Traffic-Logging-On (HOP/TRACE/R) Supervisory Message Format

	59	51		49	43 ... 0
ta	hop	0	0	notr	unused

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

hop — Primary function code $D0_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol NOTR.

Figure 3-59. Host Operator Request-to-Turn-AIP-Traffic-Logging-Off (HOP/NOTR/R) Supervisory Message Format

	59	51		49	43 ... 0
ta	hop	0	0	rel	unused

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

hop — Primary function code $D0_{16}$. You can access this field with the reserved symbol PFC, as described in section 4. Its value is the value of the reserved symbol HOP.

rel — Secondary function code $0D_{16}$. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol REL.

Figure 3-60. Host Operator Request-to-Release-Debug-Log-File (HOP/REL/R) Supervisory Message Format

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 3-61) is sent from NAM to the application program when the operator enters the K-display command:

K.RS=appname

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

	59	51		49	43 ... 0
ta	hop	0	0	rs	unused

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

hop — Primary function code $D0_{16}$. You can access this field 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. You can access this field with the reserved symbol SFC, as described in section 4. Its value is the value of the reserved symbol RS.

Figure 3-61. Host Operator Request-to-Restart-Statistics-Gathering (HOP/RS/R) Supervisory Message Format 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. The operator can select an idle-down option that permits gradual termination of operations, usually as a normal part of network service. The operator 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 3-62 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 shut-down 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 shut-down 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.

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 operator. This then permits normal termination of the network software.

Figure 3-63 shows the two forms of the host-shutdown supervisory message. The application program does not issue a response to this supervisory message.

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 3-64). The message used in this sequence is the logical-error supervisory message, shown in figure 3-65. The application program does not send a response to this supervisory message.

Application	NAM	Message
←――――――		SHUT/INSD/R (idle-down)
――――――→		CON/END/R
←――――――		CON/END/N

The application program fetches all queued upline blocks from all terminals or other application programs, then ends all connections prior to a shutdown of the network.

The application program can then disconnect from the network with a call to the AIP routine NETOFF. (See section 5.)

Application	NAM	Message
←――――――		SHUT/INSD/R (disable)

The application program must perform an immediate call to NETOFF to avoid being aborted by system console operator commands during the network shutdown in progress.

Application	NAM	Message
←――――――		SHUT/INSD/R (idle-down)
――――――→		CON/END/R
←――――――		CON/END/N
←――――――		SHUT/INSD/R (disable)

The application program fetches as many queued upline blocks as possible and ends as many connections as possible prior to shutdown of the network, then issues its NETOFF call immediately after receipt of the second shutdown message.

Figure 3-62. Host Shutdown Supervisory Message Sequences

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.

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 application block header or in a 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 application block header.

On an application-to-terminal connection, an application character type other than 2, 3, or 4 was used in a downline block header or in a change-input-character-type supervisory message.

TABLE 3-2. VALID CCP FIELD NUMBERS AND FIELD VALUES (Contd)

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes 1	Field Value Range	Field Value Content Meaning
Line feed idle count (LI)	2D (55)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 7F	Number to insert
	2F (57)	1 thru 8, 28 thru 31 (9 thru 18)	1	TIP should calculate number
Lockout unsolicited messages (LK)	20 (40)	1 thru 15, 18, 28 thru 31 (16)	0 or 1	Yes (1), no (0)
Output control (OC)	44 (104)	1 thru 3, 5 thru 8, 28 thru 31 (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Output device (OP)	36 (66)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru 2 5	Printer (0), display (1), paper tape (2)
Parity processing (PA)	32 (62)	1 thru 3, 5 thru 8, 28 thru 31	0 thru 4	Zero (0), odd (1), even (2), none (3), ignore (4)
Page waiting (PG)	25 (45)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16, 17)	0 or 1	Yes (1), no (0)
Page length (PL)	24 (44)	1 thru 18, 28 thru 31	0, 8 thru FF 5	Number of physical lines
Page width (PW)	23 (43)	1 thru 18, 28 thru 31	0, 20 thru FF	Number of characters
Site-defined use	5A thru 63 (132 thru 143)	1 thru 18, 28 thru 31	0 thru FF 5	Site-defined
Special editing mode (SE)	30 (60)	1 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Yes (1), no (0)
Terminal class (TC)	22 (42)	1 thru 10, 6 28 thru 31	01 thru 0F 5	Number of new class
Multiple-message 4 transparent delimiters (XL)	38 (70)	1 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Character specified (1), not specified (0)
Message delimiter	39 (71)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru F	Character count (upper byte)
Message delimiter	3A (72)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 thru FF	Character count (lower byte)
Message delimiter	3B (73)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31 (9, 14, 16)	0 thru FF 5	Numerical value for character

TABLE 3-2. VALID CCP FIELD NUMBERS AND FIELD VALUES (Contd)

Command (Mnemonic)	Field Number (Octal)	Usable for Terminal Classes ①	Field Value Range	Field Value Content Meaning
Mode delimiter	3C (74)	1 thru 3, 5 thru 8, 28 thru 31 (9 thru 18)	0 or 1	Timeout (1), no timeout (0)
Message delimiter	92 (222)	1 thru 3, 5 thru 8, 28 thru 31, (9 thru 18)	0 or 1	Forward on timeout (1), do not forward on timeout (0)
Mode delimiter	45 (105)	1 thru 8, 28 thru 31 (9 thru 18)	0 thru FF ⑤	Numerical value for character
Mode type	46 (106)	1 thru 8, 10, 13, 15, 28 thru 31	1	Multiple-message (1)
Full duplex (none)	57 (127)	1 thru 3, 5 thru 8, 28 thru 31 (4, 9 thru 18)	0 or 1	Yes (1), no (0)
Terminal transmission block size (none)	1E (36)	1 thru 18, ⑥ 28 thru 31	0 thru 7	Number of characters (upper byte)
	1F (37)	1 thru 18, ⑥ 28 thru 31	0 thru FF	Number of characters (lower byte)
Set terminal in solicited input mode	70 (160)	1 thru 8, 10 thru 13, 15, 18, 28 thru 31	0 or 1	Yes (1), no (0)
Carriage return idle delay	93 (223)	1 thru 8, 28 thru 31, (9 thru 18)	0 thru FA	Idle delay in increments of 4 milliseconds
Linefeed idle delay	94 (224)	1 thru 8, 28 thru 31, (9 thru 18)	0 thru FA	Idle delay in increments of 4 milliseconds

Notes:

① No error occurs if an FN/FV pair is issued for a terminal class shown in parentheses.

② Ignored for CDC-defined X.25 packet assembly/disassembly (PAD) terminals.

③ Any hexadecimal value except 00 thru 02, 20, 30 thru 39, 3D, 41 thru 5A, 61 thru 7A, or 7F.

④ If the value of one of the fields for this command is changed, you need to ensure that the others are set to known values if they could affect your application. All of the fields need not be specified. However, any fields not specified contain their previously recorded setting which could produce undesirable results.

⑤ Not all values are legal for all terminal classes.

⑥ Not allowed for CDC-defined X.25 packet assembly/disassembly (PAD) terminals. For terminal class (TC) changes, refer to Effects of Changing Terminal Class on CDCNET, in this section.

	16 thru 256	Reserved for the NAM subsystem.

You can access this field with the reserved symbol RC, as described in section 4.

abherr — Application block header word associated with the supervisory message that caused the ERR/LGL/R message. This field contains a non-zero word unless the rc value is 7. You can access this field with the reserved symbol ERRABH, as described in section 4.

firstwrd — The first 60 bits of the 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 non-zero word unless the rc value is 7. You can access this field with the reserved symbol ERRMSG, as described in section 4.

Figure 3-65. Logical-Error (ERR/LGL/R) Supervisory Message Format (Sheet 2 of 2)

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 independent 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 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 AIP statement calling sequence does not permit recursive calls.

PREDEFINED SYMBOLIC NAMES

The fields in NAM supervisory messages of application character types 1 and 2 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.

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 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 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 call to store the information is:

```
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.

```
[label1] macro-name {LIST=label2        }
                    {LIST=register name }
```

When this format is used, macro expansion assumes that the proper calling parameter block is located at the address specified by the LIST value, loads this address into register A1, and performs the call to the AIP procedure.

label2 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. The second and third forms 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.

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
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
ABHNCP	No cursor positioning flag bit in application block header for downline (output) blocks.	None
ABHNEP	No echoplex flag bit in application block header for downline (output) 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
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	Application character type of CON supervisory messages, for use in application block header	1
ACCTRL	Application character type of CTRL supervisory messages, for use in application block header	2
ACDBG	Application character type of DBG supervisory messages, for use in application block header	1
ACDC	Application character type of DC supervisory messages, for use in application block header	1
ACERR	Application character type of ERR supervisory messages, for use in application block header	1
ACFC	Application character type of FC supervisory messages, for use in application block header	1
ACHOP	Application character type of HOP supervisory messages, for use in application block header	1
ACIFC	Application character type of IFC supervisory messages, for use in application block header	1

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
ACINTR	Application character type of INTR supervisory messages, for use in application block header	1
ACK	Secondary function code field for FC/ACK/R	2
ACLST	Application character type of LST supervisory messages, for use in application block header	1
ACRQ	Secondary function code field for CON/ACRQ messages	2
ACSET	Application character type of SET supervisory messages, for use in application block header	1
ACSHUT	Application character type of SHUT supervisory messages, for use in application block header	1
ACTCH	Application character type of TCH supervisory messages, for use in application block header	1
ALT	Secondary function code field in HOP/ALT/R	B
APP	Secondary function code field for INTR/APP/R	2
BI	Primary function code field for BI/MARK/R	CA_{16}
BIMARK	Primary and secondary function code fields for BI/MARK/R, including EB and RB fields as zero	$CA00_{16}$
BRK	Secondary function code field for FC/BRK/R and HOP/BRK/R	0
CB	Secondary function code field for CON/CB/R	5
CCD	Secondary function code field for CON/CCD/R	$0C_{16}$
CHAR	Secondary function code field for CTRL/CHAR/R	8_{16}
CICT	Secondary function code field for DC/CICT/R	0
CMD	Secondary function code field in HOP/CMD/R	1
CON	Primary function code field for connection management (CON) supervisory messages	63_{16}
CONAABL	Application block limit field in CON/ACRQ/R	None
CONABN	Application block number field of CON/REQ/R	None
CONAABN	Application block number field of CON/ACRQ/R	None
CONAAWC	User validation control word in CON/REQ/R	None
CONABL	Application block limit field in CON/REQ/R	None
CONABN	Application block number field of CON/ACRQ/R	None
CONABZ	Block size in connection management (CON) supervisory messages	None
CONACN	Application connection number field in connection management (CON) supervisory 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 function code fields in CON/ACRQ/A including EB field set to 1	6382_{16}

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
CONACT	Application input character type field in CON/REQ/N	None
CONADBL	Downline block limit field in CON/ACRQ/R	None
CONADBZ	Downline block size field in CON/ACRQ/R	None
CONAHDS	User validation control word in CON/REQ/R	None

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
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 CON/REQ/R	None
CONDT	Device type field in CON/REQ/R	None
CONEND	Primary and secondary function code fields in CON/END/R, incluoing 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}
CONFAM	Login family name field in CON/REQ/R	None
CONFO	Login family ordinal field in CON/REQ/R	None
CONHID	Host node field in CON/REQ/R	None
CONICT	Application input character type field in CON/REQ/N	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	Subdevice 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 CON/REQ/R	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	$C108_{16}$

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
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 (CTRL) supervisory messages	$C1_{16}$
CTRRTC	Primary and secondary function code fields for CTRL/RTC/R, including EB and RB fields as zero	$C109_{16}$
CTRTCD	Primary and secondary code fields in CTRL/CHAR/R, including EB and RB fields as zero	$C10A_{16}$
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 field in DC/CICT/R	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 character 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
ENDD	Secondary function code field in CON/END/R	6
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 (FC) supervisory messages	83_{16}
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 (FC) supervisory messages	None
FCBRK	Primary and secondary function code fields in FC/BRK/R, including EB and RB fields as zero	8300_{16}
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}

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
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}
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 host operator (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 (INTR) supervisory messages	80_{16}
INTRACN	Application connection number field in user-interrupt (INTR) supervisory messages	None
INTRAPP	Primary and secondary function code fields in INTR/APP/R, including EB and RB fields as zero	8002_{16}
INTRCHR	Field containing ASCII alphabetic character A through Z in typeahead priority data 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 supervisory 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

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
LCONENN	Length in 60 bit words of CON/END/N	1
LCONREQ	Length in 60-bit words of CON/REQ/R message	10 (A_{16})
LCORQR	Length in 60-bit words of CON/REQ/N and CON/REQ/A	1
LCTRL	Length in 60-bit words of terminal control (CTRL) supervisory messages	2
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 (FC) supervisory messages (except FC/BRK)	1
LFCACK	Length in 60-bit words of FC/ACR/R	1
LFCBRK	Length in 60-bit words of FC/BRK/R	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	1
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 (LST) supervisory messages	1
LSHUT	Length in 60-bit words of SHUT/INSD/R	1
LST	Primary function code field in list management (LST) supervisory messages	$C0_{16}$
LSTACN	Application connection number field in list management (LST) supervisory messages	None
LSTALN	Application list number field in list management (LST) 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	$C004_{16}$

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
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	$C002_{16}$
LTCH	Length in 60-bit words of TCH/TCHAR/R	1
MARK	Secondary function code field in TO/MARK/R, BI/MARK/R, and RO/MARK/R	0
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 supervisory 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
RO	Primary function code field in RO/MARK/R	CB_{16}
ROMARK	Primary and secondary function code fields in RO/MARK/R, including EB and RB fields as zero	$CB00_{16}$
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	Shutdown type 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}

TABLE 4-1. RESERVED SYMBOLS (Contd)

Symbol	Entity Defined by Symbol	Predefined Integer Value
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
TO	Primary function code field in TO/MARK/R	$C4_{16}$
TOMARK	Primary and secondary function code fields in TO/MARK/R, including EB and RB fields as zero	$C400_{16}$
TRACE	Secondary function code field in HOP/TRACE/R	2
USR	Secondary function code field in INTR/USR/R	0

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.

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 ≤ 18 bits long.

Figure 4-1. NFETCH Macro Call Format

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.

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.

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
```

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

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. Appendix B contains a complete listing of the assembly error messages from NSTORE.

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 used when NAM is used by a COBOL program for input and output between the program and terminals. 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 section 8. 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.

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.

[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

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 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 5 statement:

```
M=NFETCH(0,L"CON")
```

makes M the integer value 143_8, 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 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.

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

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.)

As examples of NSTORE use, consider the following operations.

Example 1:

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 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 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.

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.

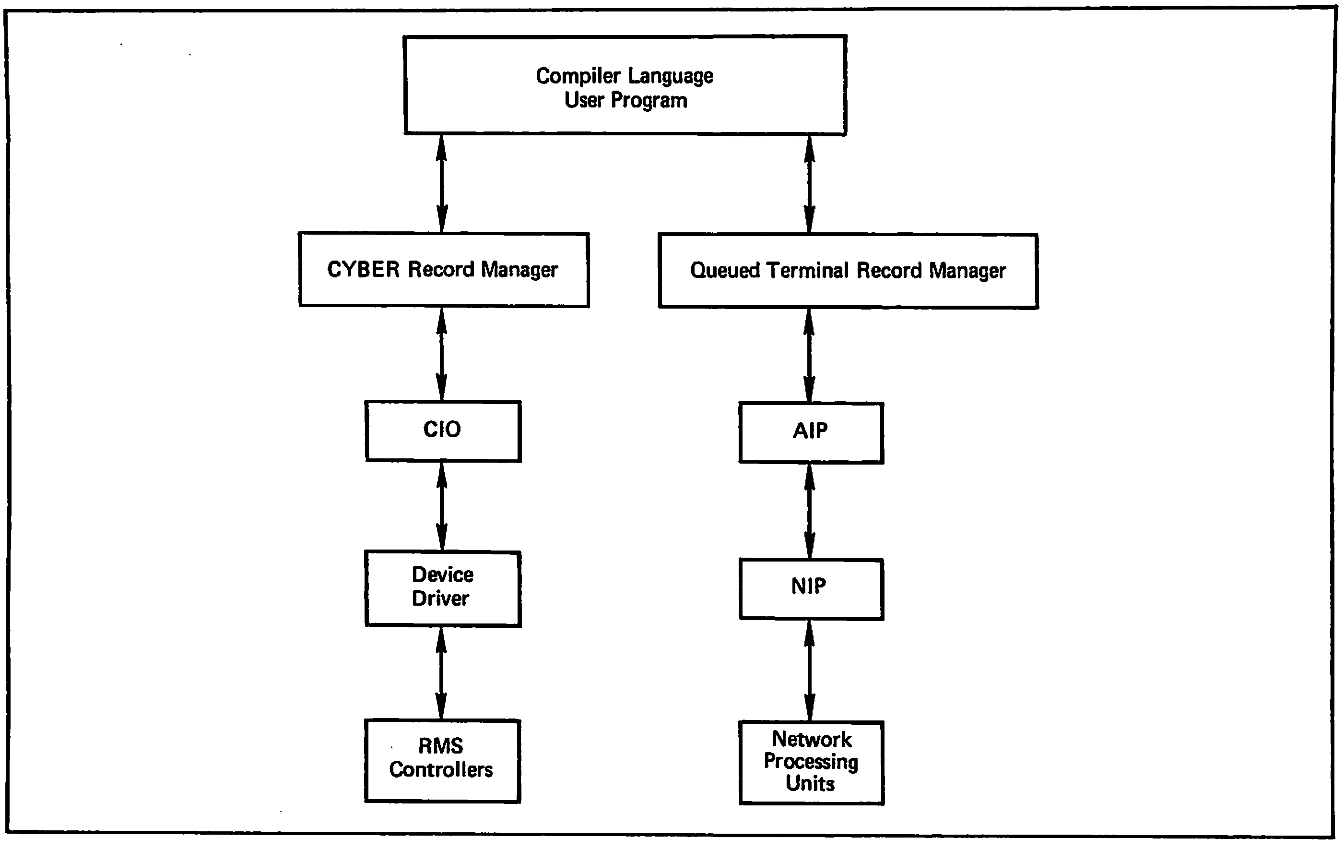


Figure 4-5. QTRM Interface Level Analogy

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 end 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

Half-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

Application supplied OUTCALL parameters for application-to-application connections sending or receiving user data during the establishment of application-to-application connections

Sending a break (FC/BRK) or INTR/APP message

Qualified data as described in section 2

Logical identifiers (LIO's) in the establishment of application-to-application connections

Section 8 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, programmers with a need for understanding communication among the components of the network software, and programmers needing to interpret a load map.

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 or data messages, and to request a supervisory or incoming (upline) data message. 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 NETWAIT call with a flag value of 0 and there are supervisory messages or data available 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. 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

A complete compiler or assembler listing for an application program contains 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 an internal block protocol not described in section 2. These blocks are compiled or interpreted by NIP.

TABLE 4-2. AIP INTERNAL PROCEDURES

Name	Function
NP$CLK	Used only when AIP is run with either the debugging or statistics option on; gets system clock time.
NP$DATE	Used only when AIP is run with either the debugging or statistics option on; gets current date.
NP$DBG	Used only when AIP is run with the debugging option on; makes entries in the debug log file (application program local file ZZZZZDN). These entries show results of calls to other AIP routines by the program. (See section 6.)
NP$DMB	Dumps field length to the application program local file ZZZZDMB.
NP$ERR	Issues error messages to the application program's dayfile.
NP$GET	Creates NETGET, NETGETL, NETGETF, or NETGTFL worklist entry to send to NIP.
NP$GSM	Refills AIP's supervisory message buffer. (See Worklist Processing.)
NP$MSG	Issues dayfile message to NIP's dayfile.
NP$ON	Processes NETON call response from NIP.
NP$OSIF	Issues system control point (SSC) RA+1 call.
NP$PUT	Creates NETPUT worklist entry to send to NIP.
NP$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$SN	Used only when AIP is run with the statistics option on; accumulates statistical data.
NP$SPRT	Used only when AIP is run with the statistics option on; makes entries in the debug log file (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 supervisory status word. (See section 5.)
NP$WRTO	Used only when AIP is run with the debugging option on; writes one word in the debug log file (application program local file ZZZZZDN). (See section 6.)
NP$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. (See section 6.)
NT$WRTW	Used only when AIP is run with either the debugging or statistics option on; writes entry to the debug log file or statistics file. (See section 6.)
NP$XCDD	Used only when AIP is run with the statistics option on; converts numbers to decimal form in 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$GETS	Controls variables used to process NETGET, NETGETL, NETGETF, and NETGTFL calls.
NP$LOF	Used only when AIP is run with the debugging option on; parameter block for SETLOF macro. (See section 6.)
NP$MODE	Used to keep track of the state the application is in.
NP$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$TAA	Used to reference the text area array (TAA) in fragmented NETGETF and NETPUTF or NETGTFL calls.
NP$ZHDR	Header entry for the debug log file (application program local file ZZZZZDN).

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
```

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 requirements for such to be successfully validated. (See section 6.)

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 6-bit display code the name of the application program, as it is identified for log in and for CONTNAP. This can be one to seven 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	LOGIN	NUL	PTFS	TCF
BYE	LOGOUT	NVF	QTFI	TVF
CS	MCS	PFU	QTFS	
HELLO	NAM	PNI	RBF	
IAF	NIP	PSU	RMF	
ITF	NS	PTFI	TAF	

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 the maximum number of allowed applications has already netted on.

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 host operator has enabled it.

minacn An input parameter, specifying the smallest application connection number the application program can process; $0 < \text{minacn} \leq \text{maxacn} \leq 4095$. 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; $0 < \text{minacn} \leq \text{maxacn} \leq 4095$. 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

	59		57		55	54	53	29 0
nsup	c	a	n	i	s	d	res	mc

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 NETCHEK 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 system that NAM is no longer available.

i Input-in-queue bit. This bit is set to one if NIP has either data messages or 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.

d Data-deliverable bit. This bit is set to one if data messages are deliverable on at least one of the connection lists of the application program and the application program issues a NETGETL or a NETGTFL call.

res Reserved for CDC. Reserved fields contain zero.

mc A count of the number of supervisory messages and network data blocks 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 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 RMV2, is identified by that name in COMTNAP and in the local configuration file as a non-privileged application. RMV2 can process up to three logical connections but requires connections to be numbered beginning with 2. RMV2 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.

```
      COMMON NSUP,HA(2),TA(200,2)
        •
        •
      NAME=4HRMV2
      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 FORTRAN Example

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

```
CALL NETOFF
```

Figure 5-4. NETOFF Statement FORTRAN Call Format

The NETOFF statement is used 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 device 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).

NETWORK 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)

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 network data blocks from application connection numbers other than 0. Synchronous supervisory messages and network data blocks are never queued on logical connection 0.

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 network block in the application program's text area. The NETGET statement has the format shown in figure 5-5.

CALL NETGET(acn,ha,ta,tlmax)

acn — An input parameter, specifying the application connection number of the logical connection from which a block is requested. This parameter can have the values:

0	Transfer one asynchronous supervisory message.
minacn ≤ acn ≤ maxacn	Transfer one network 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.

Figure 5-5. NETGET Statement FORTRAN Call Format (Sheet 1 of 2)

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 ≤ 410 for 60-bit (one per word) transparent characters act=2 $1 \leq$ tlmax ≤ 273 for 8-bit (7.5 per word) ASCII characters act=3 $1 \leq$ tlmax ≤ 410 for 8-bit (5 per word) ASCII characters act=4 $1 \leq$ tlmax ≤ 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 (Sheet 2 of 2)

If no network 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 network block larger than this size is queued from the specified connection, the network 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 network 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 truncated. (See section 3.) If block truncation is in effect at the time of the NETGET call, then the block is 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.

```
      •
      •
      INTEGER TA(26),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
      •
      o
      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.26) GO TO 9
      CALL NETGET(NACN,HA,TA,OVTLMAX)
      GO TO 6
    4 CONTINUE
      •
      •
    9 STOP
```

Figure 5-6. NETGET Statement FORTRAN 5 Examples

Figure 5-6 contains two examples of NETGET use. The first occurrence is in fetching asynchronous connection-request supervisory 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 device 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 thirteen 60-bit characters and for a logical line of 72 teletypewriter characters, or for a minimum-sized network block of 100 characters from a longer logical line, 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).

Inputing to Fragmented Buffer Array (NETGETF)

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 network data blocks from application connection numbers other than 0. Synchronous supervisory messages and network data blocks are never queued on logical connection 0.

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 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 5-7.

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 5-8.

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 block larger than this total size is queued from the specified connection, the 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.

```
CALL NETGETF(acn,ha,na,taa)
```

acn — An input parameter, specifying the application connection number of the logical connection from which a block is requested. This parameter can have the values:

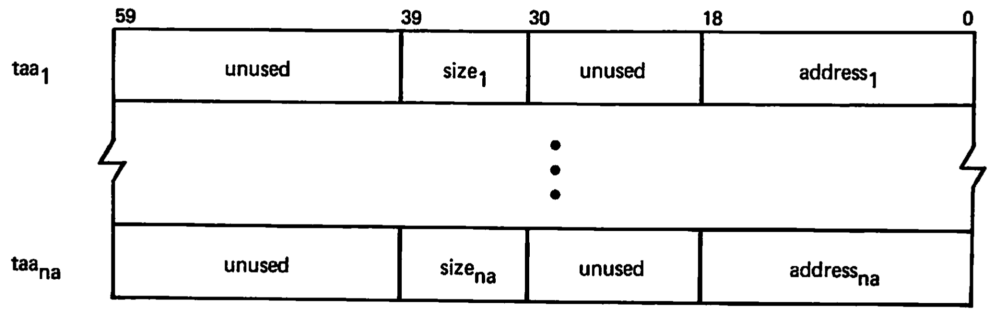
0	Transfer one asynchronous supervisory message.
minacn $\le$ acn $\le$ maxacn	Transfer one network data block or synchronous supervisory message from the logical connection with the indicated acn.

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 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 $\le$ na $\le$ 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 5-8.

Figure 5-7. 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 $1 \le size_i \le 63$. 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 5-8. NETGETF Statement Text Area Address Array

Figure 5-9 contains 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.

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.

```
      •
      •
   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 5-9. NETGETF Statement FORTRAN 5 Examples

Outputing From Single Buffer (NETPUT)

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 network data blocks to application connection numbers other than 0. Synchronous supervisory messages and network data blocks are never sent on logical connection 0.

Each NETPUT call requests AIP to form a block from the information located in the application program's block header and text areas. The calling application program must construct a complete block header, as described in section 2. The text portion of the block can be either a network 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 5-10.

```
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 network data or super-
        visory message block, correctly described
        by the contents of the block header area.
```

Figure 5-10. 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.)

Figure 5-11 contains 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 5-11. NETPUT Statement
FORTRAN 5 Example

Outputing From Fragmented Buffer Array (NETPUTF)

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 network data blocks to application connection numbers other than 0. Synchronous supervisory messages and network data blocks are never sent on logical connection 0.

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 block header, as described in section 2. The text portion of the block can be either a network 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 5-12.

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 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 $1 \leq na \leq 40$.

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 5-13.

Figure 5-12. NETPUTF Statement FORTRAN Call Format

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. (The text area address array is shown in figure 5-13.) 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

	59	39	30	18 0
taa_1	unused	$size_1$	unused	$address_1$
	⋮			
taa_{na}	unused	$size_{na}$	unused	$address_{na}$

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 $1 \leq size_i \leq 63$. The sum of all na values for $size_i$ 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 5-13. NETPUTF Statement Text Area Address Array 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.)

Figure 5-14 contains 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 7-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 seven words of ten 6-bit characters each, a size of 7 is declared in each OTAA entry.

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)

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 network data blocks on connection numbers other than zero can also be obtained when their connection numbers have been assigned to list 0.

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 network data blocks queued and that has not been turned off by a LST/OFF/R supervisory message. One network data 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 block body in the application's text area. Figure 5-15 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.

```
       •
       •
   DIMENSION OLINE1(7),...,OLINE12(7)
   INTEGER HA,OTAA(12),ONA,TERM(20)
   DATA ONA/12/,HA/0/,OLINE1/"ABCDEFGHIJ",...,L"12345678",0/,...,
  1DATA OLINE12/"ABCDEFGHIJ",...,L"12345678",0/
       •
       •
   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",840)
   CALL NETPUTF(HA,ONA,OTAA)
       •
       •
```

Figure 5-14. NETPUTF Statement FORTRAN 5 Example

```
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 ≤ 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 ≤ 410 for 60-bit (one per word) transparent characters
act=2	$1 \leq$ tlmax ≤ 273 for 8-bit (7.5 per word) ASCII characters
act=3	$1 \leq$ tlmax ≤ 410 for 8-bit (5 per word) ASCII characters
act=4	$1 \leq$ tlmax ≤ 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 5-15. NETGETL Statement FORTRAN Call Format

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.

The application program indicates the size of its buffer in each NETGETL call. If a block larger than this size is available for transfer, the block remains queued, unless data 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 block 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.

Figure 5-16 contains 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

```
      •
      •
   INTEGER TA(26),HA,TLMAX,OVTLMAX
   DATA HA/0/,TA/26*0/,TLMAX/13/
      •
      •
  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.26) GO TO 9
   NACN=NFETCH(HA,L"ABHADR")
   CALL NETGET(NACN,HA,TA,OVTLMAX)
      •
      •
   GO TO 1
 5 CONTINUE
      •
      •
 9 STOP
```

Figure 5-16. NETGETL Statement FORTRAN 5 Example

batch connections. The TLMAX value of 13 is adequate for both supervisory messages of application character type 1 and 72-character logical lines or a minimum-sized network block of 100 characters 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)

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 network data blocks on connection numbers other than zero can be obtained when their connection numbers have been assigned to list 0.

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 blocks queued and has not been turned off by a supervisory message. One 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 body is placed in the application's text areas. Figure 5-17 shows the format of the NETGTFL statement.

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.
- **$1 \le aln \le 63$** — 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 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 $1 \le na \le 40$.

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 5-18.

Figure 5-17. NETGTFL Statement FORTRAN Call Format

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.

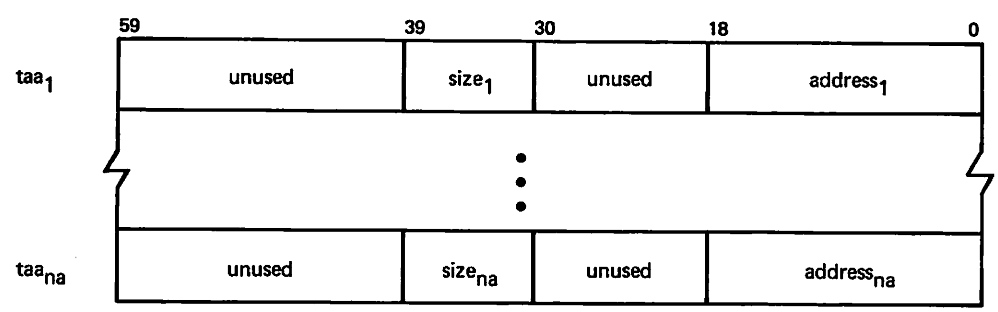
The text areas used are defined for AIP by the text area address array identified in the NETGTFL call. This text area address array has the format shown in figure 5-18.

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 block larger than this total size is queued from the specified connection, the 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 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.

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.

Figure 5-19 contains 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 network data 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.



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 5-18. NETGTFL Statement Text Area Address Array

```
     •
     •
  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 5-19. NETGTFL Statement FORTRAN 5 Example

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.

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 5-20) 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

CALL NETWAIT(time,flag)

time — An input parameter, $1 \leq time \leq 4095$, 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 connection, 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 5-20. NETWAIT Statement FORTRAN Call Format

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 zero 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.

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.

Figure 5-21 contains 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 5-21. NETWAIT Statement FORTRAN 5 Examples

CONTROLLING PARALLEL MODE (NETSETP)

The NETSETP statement (figure 5-22) 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.

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 5-22. NETSETP Statement FORTRAN Call Format

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.

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.

Figure 5-23 contains 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 a 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 5-23. NETSETP and NETCHEK Statement FORTRAN 5 Examples

The program flow shown minimizes worklist transfers by concentrating the console 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 5-24) 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 5-24. 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. Figure 5-23 contains examples of NETCHEK use.

This section describes the structure and execution of a Network Access Method (NAM) application program.

NOTE

You cannot execute application programs as Transaction Facility tasks.

NOS SYSTEM CONTROL POINT FACILITY

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 drops to zero. The GETACT macro is described in volume 4 of the NOS reference set.

BATCH 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 user job 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.

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 command.

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 to define the symbolic names used for the field access macro utilities NFETCH and NSTORE. (See section 4.)

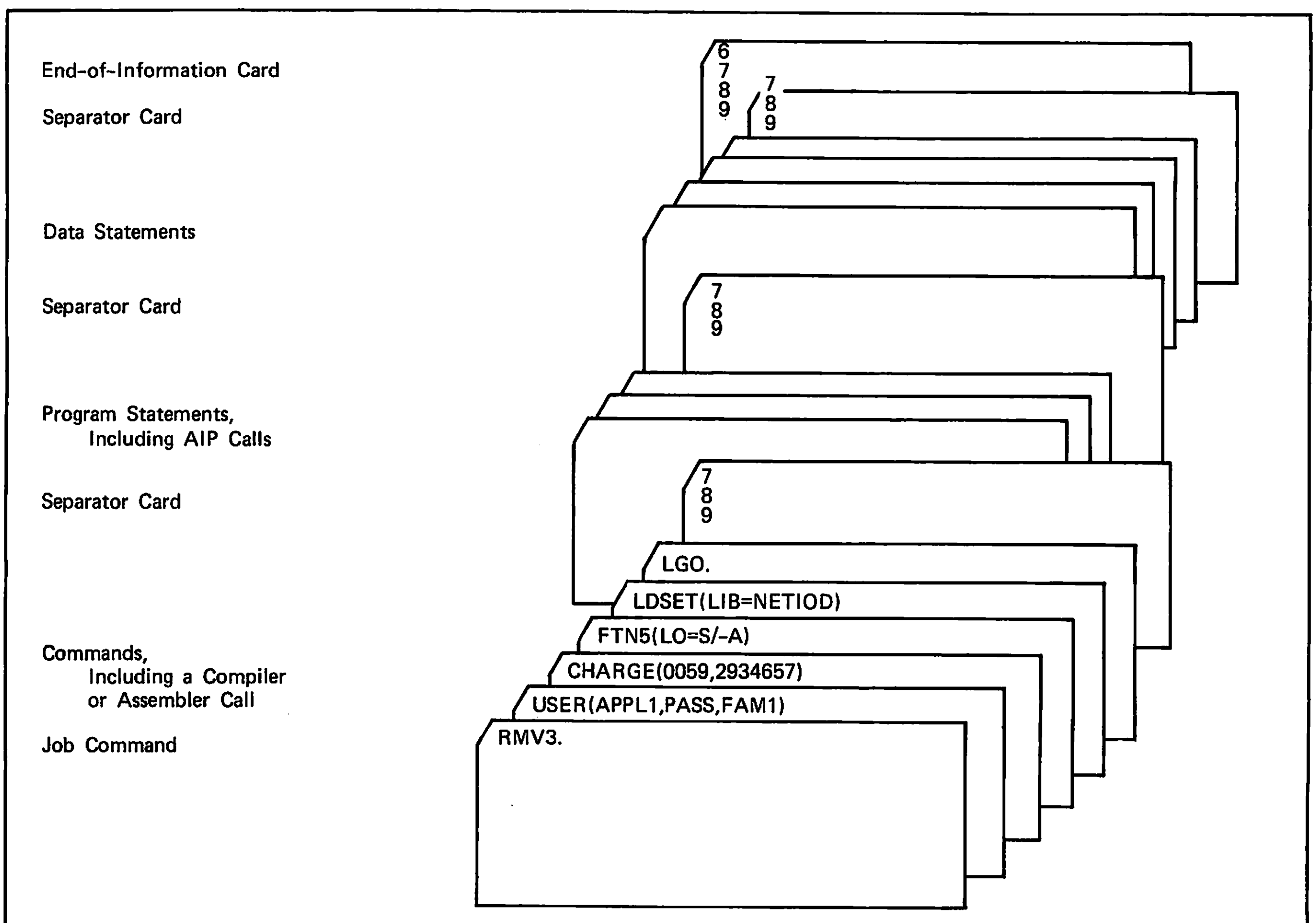


Figure 6-1. Typical Job Structure for System Input

JOB IDENTIFICATION

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 host 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 host 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.

PROGRAM CONTENT

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 a new NETON call in order to continue accessing the network through NAM.

When an application program 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.

PROGRAM EXECUTION THROUGH IAF

Your application program can be executed from the Interactive Facility in several ways:

- As a SUBMIT command file batch job
- As a ROUTE command file batch job
- As an interactive job
- As a detached interactive job (so your terminal can log in to it)

The use of SUBMIT and ROUTE is described in volume 3 of the NOS reference set. SUBMIT and ROUTE command file jobs have the same command content requirements as other batch jobs.

Figure 6-2 shows the procedure for interactive execution of the sample program RMV2 (chapter 8). Detached interactive job programs have the same program content requirements as batch job programs.

```
Your entries are underlined:

/attach,rmv <------------------------------ Attach direct access source file
/ftn5,i=rmv,lo=0,b=zap <------------------- Compile it
    0.479 CP SECONDS COMPILATION TIME.
/ldset(lib=netiod <------------------------ Load it
LDR>? zap <-------------------------------- Execute it
ESCe <------------------------------------- Bypass the IAF input queue to find out if the job step
                                            was successful

 JSN: AAYS  SYSTEM: BATCH    SRU:       4.889
 STATUS:  NAM  VER 1.5-  2D
ESCd <------------------------------------- Detach the running (rolled out) application program

  JOB DETACHED, JSN=AAYS
JSN: AAZB, NAMIAF

   RECOVERABLE JOB(S)

   JSN    UJN       STATUS       TIMEOUT

   AAYS   AANY      EXECUTING
```

Figure 6-2. Interactive Program Execution Procedure Example (Sheet 1 of 2)

```
ENTER  GO  TO CONTINUE CURRENT JOB,
       RELIST  TO LIST RECOVERABLE JOBS,
       OR DESIRED JSN:  go <--------------- Startup a new job so you can switch applications

/bye,rmv2 <-------------------------------- Use an IAF application switch command

UN=XXXXXXX  LOG OFF  12.07.08.
JSN=AAZB     SRU-S      2.003
IAF     CONNECT TIME 00.04.01.

RMV2 VER 3
INPUT PLSSHUTD <--------------------------- Respond to RMV2 prompt with command that shuts it down

RMV2    CONNECT TIME 00.00.08.

JSN: AAZC, NAMIAF <------------------------ Connection switch back to IAF is automatic

   RECOVERABLE JOB(S)

   JSN    UJN       STATUS       TIMEOUT

   AAYS   AANY      SCP ROLLED

ENTER  GO  TO CONTINUE CURRENT JOB,
       RELIST  TO LIST RECOVERABLE JOBS,
       OR DESIRED JSN:  aays <------------- Recover the detached application program (has called
                                            NETOFF, so this rollout is controlled by IAF)

 JSN: AAYS  SYSTEM: BATCH    SRU:      0.034
 STATUS:
 CHARACTER SET: NORMAL   MODES: PROMPT ON
 JOB IN SYSTEM.  ENTER GO TO CONTINUE.
go <--------------------------------------- Roll it back in
ACSR,      1.000UNTS.
/enquire,f <------------------------------- Here are all the files NETIOD should create

LOCAL FILE INFORMATION.

 FILENAME  LENGTH/PRUS  TYPE  STATUS    FS

  INPUT*          1     IN.*  BOI
  INPUT                 LO.
  OUTPUT                LO.
  ZZZZZDN         3     LO.   EOR WRITE
  SUBFILE         1     LO.   BOI
  RMV            34     PM.*  EOR
  ZAP            32     LO.   EOF
  ZZZZZSN         2     LO.   EOR WRITE

 TOTAL = 8
```

Figure 6-2. Interactive Program Execution Procedure Example (Sheet 2 of 2)

TYPES OF APPLICATION PROGRAMS

All application programs should be specified in COMTNAP. When an application is defined also in the local configuration file it can be declared as having one of the following attributes:

- Disabled
- Unique identifier
- Privileged
- Request startable
- Have more than one copy on any one host

Access to an application program can also be controlled. A program can be:

- A restricted access or general access application program
- A mandatory or primary application program

These access types are separately established for each connection with the program. The first type is controlled through the user name associated with the connection. The second type is controlled through the terminal device name associated with the connection.

DISABLED

A disabled application is configured in the network but is not allowed to access the network until the host operator enters an enable command to allow it to be connected.

UNIQUE IDENTIFIER

A unique identifier application program requires that interactive console user access to it be restricted on the basis of the login parameters used. Only one interactive console with a given combination of family name and user name 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 name and user name index combination used by a console 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 name and user name index combinations of all consoles accessing the program are guaranteed to be unique.

PRIVILEGED

Privileged application programs must have an SSJ= entry point to access the network successfully. They also often have the CSOJ bit set in the access word associated with the user name for the job executing the program code.

The CSOJ bit provides the program with system origin type permission. Jobs with system origin type permission can be executed by host 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 but do not contain an SSJ= entry point. 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 usually become 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 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 with the CSOJ bit set has not been installed in the system library, it can be executed by a host 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 a 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 with the CSOJ bit set can be executed by a host 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 with the CSOJ bit set 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 system 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.

Privileged application programs with the CSOJ bit set can be automatically started when the host's network software is started. This process is described in the NOS Administration reference manual.

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, remote batch, or detached interactive job origin during the debugging process.

REQUEST STARTABLE

Whenever the application is requested by a terminal user (through the application name in the login process), or by another application (by a CON/ACRQ message), NVF attempts to start the application.

The file name equivalent to the name of the application should be made available to NVF through the NVF startup record. (See the NOS Installation Handbook.)

HAVE MORE THAN ONE COPY (ON ANY ONE HOST)

More than one copy of an application program is allowed to be simultaneously connected to the network, if so specified in the local configuration file. If such an application is also request startable, then NVF will start up a new copy of an application whenever a connection request for the application comes into the host, and all existing copies already have their maximum number of connections.

RESTRICTED OR GENERAL ACCESS

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 Administration reference manual.

MANDATORY OR PRIMARY

In the local configuration file, each terminal console 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 entered during the login. For subsequent connections, the network will prompt for an application and, if a carriage return is entered, the network will connect the terminal to the primary application.

DEBUGGING APPLICATION PROGRAMS

Application program job content partially depends 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 command, 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 network data blocks 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 6-3 shows the NETDBG utility FORTRAN call statement format. NETDBG can only be called after NETON is called and before NETOFF is called.

Calls to NETDBG can occur in programs using either NETIO or NETIOD. For example, when a NETDBG call turns either or both supervisory message and network data block 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 6-4 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. NETREL reads up to 192 central memory words of the named file, or until a logical end-of-record is encountered.

The second parameter on the NETREL call gives the maximum number of words in each message to be saved in the ZZZZZDN file.

```
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 network data block logging on.
- ≠0 — Turn network data block logging off.

When network data block logging is turned on, all network data blocks exchanged on any connection between the application program and NAM are logged; block-delivered supervisory messages (FC/ACK/R) are also logged, regard less of the value specified for the dbugsup parameter. Logging occurs whenever a call to NETGET, NETGETL, NETGETF, NETGTFL, NETPUT, or NETPUTF causes a block 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 6-3. NETDBG Utility FORTRAN Call Statement Format

```
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 command
          record file before the NETREL oper-
          ation 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 6-4. NETREL Utility FORTRAN Call Statement Format

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 MC field in the supervisory status word (NETON nsup parameter). Subsequent calls to NETREL route ZZZZZDN to the input queue, reinitialize the file environment table and MC field 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.

NETSETF Utility

NETSETF calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5.) Figure 6-5 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 23 octal (the COMPASS EREXIT mnemonic 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.

```
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 re-
                  turned 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 re-
          turned by NAM. If zero, either the
          flush parameter was nonzero or NETIO
          was loaded (in which case the flush
          parameter makes no difference).
```

Figure 6-5. NETSETF Utility FORTRAN Call Statement Format

The SETLOF macro provides NOS 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.

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 6-6 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.

```
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 6-6. NETLOG Utility FORTRAN Call Statement Format

Formatted messages are stored as 6-bit display code characters with zero byte terminators. The first character of the message is used as a carriage control character for the line and is not printed. Formatted messages longer than 136 characters should be stored as separate zero-byte-terminated lines.

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 6-7 shows the NETDMB utility FORTRAN call statement format. NETDMB allows an application to dump its exchange package and central memory field 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 Analysis Handbook. A logical end-of-record is written to the file ZZZZDMB after each NETDMB call.

```
CALL NETDMB(dumpid,ecs)

dumpid    An input parameter that is an octal
          6-digit dump identifier number. The
          dumpid parameter can have the values
          0 ≤ dumpid ≤ 777777.

ecs       An input parameter that determines
          whether the associated extended
          central storage is also dumped. This
          parameter can have the values:

            =0   Do not dump extended central
                 storage

            ≠0   Dump the associated extended
                 central storage
```

Figure 6-7. NETDMB Utility FORTRAN Call Statement Format

Debug Log File Postprocessor Utility

The debug log file is a binary compressed file; 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 command is shown in figure 6-8. Examples of DLFP commands are shown in figure 6-9.

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 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 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 6-8. DLFP Command General Format

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 command and separate DLFP directives.

An optional directive file can be submitted to the DLFP to select special supervisory messages or network data blocks 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 supervisory messages or network data blocks 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 6-10 shows the general format of DLFP directive keywords with examples of them in figure 6-11.

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 6-9. DLFP Command Examples

Each directive record initiates an independent search. An empty directive file or empty directive record or I=0 causes all debug log file blocks 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 lists all dayfile messages issued by DLFP. Errors or informative messages can be writtento 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 DLFP command.

The general format of a log file listing is shown in figure 6-12. (Section 7 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 supervisory messages or blocks are logged with their declared parameters, followed by the block header contents and block text area 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 block or 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 or blocks that were logged on or after this date are output. Messages or blocks 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; $00 \le yy \le 99$, $01 \le mm \le 12$, $01 \le dd \le 31$.
BT=	hhmmss	Specifies that only messages or blocks that were logged on or after this time are output. Messages or blocks before this time are not output. If the debug log file contains more than one day's traffic, messages or blocks 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; $00 \le hh \le 24$, $00 \le mm \le 59$, $00 \le ss \le 59$.
C		Specifies that only network data blocks with the cancel flag set in the application block header are output.
CN=	n	Specifies that only synchronous and asynchronous supervisory messages and network data blocks relating to connection number n are output; $1 \le n \le 255$.
DN=		Reserved for CDC use.
E		Specifies that only supervisory messages with the error bit set are output.
ED=	yymmdd	Specifies that messages or blocks 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; $00 \le yy \le 99$, $01 \le mm \le 12$, $01 \le dd \le 31$.
ET=	hhmmss	Specifies that messages or blocks on or after this time are not to be output. If the debug log file contains more than one day's traffic, 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; $00 \le hh \le 24$, $00 \le mm \le 59$, $00 \le ss \le 59$.
LE=	n	Specifies maximum length in central memory words of each message or block to be output; $1 \le n \le 410$ (default=10).
F		Specifies that only network data blocks with the no format effector bit set in the application block header are output.
N		Specifies that only supervisory messages or network data blocks are output. Messages generated by applications for the debug log file are ignored.
NM=	n	Specifies that only n messages or blocks are output; $0 \le 1000000$.
P=		Specifies that only network data blocks 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; $00 \le hh_{16} \le FF$.
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. $0000 \le hh_{16} \le FFFF$.
R		Specifies that only supervisory messages with the response bit set are output.
SM=	n	Specifies that no messages or blocks are output until the nth message, which satisfies all the other keyword options, is found; $0 \le n \le 1000000$.
SN=		Reserved for CDC use.
T		Specifies that only upline messages or blocks with the data truncation flag bit set in the application block header are output.

Figure 6-10. DLFP Directive Keyword Format (Sheet 1 of 2)

Keyword†	Value	Description
U		Specifies that only messages or blocks with the input block undeliverable bit set in the application block header are output.
X		Specifies that only messages or blocks 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 DLFP command. 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 6-10. 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 errors. 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 errors. 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 network data blocks for 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 and blocks 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 are 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 errors. 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 6-11. 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    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
                                            .
                                            .
        nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa

hh.mm.ss.mil NETLOG (oooooo)                                                            MSG NO. ddd

        001   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
        003   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc 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    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
                                            .
                                            .
        nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc 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    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
    FRAGMENT     2    SIZE = dddd  ADDRESS = oooooo
                                            .
                                            .
    FRAGMENT     dd   SIZE = dddd  ADDRESS = oooooo
        nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc 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    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
                                            .
                                            .
    FRAGMENT     dd   SIZE = dddd  ADDRESS = oooooo
        nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc 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    oooooooooooooooooooo     cccccccccc aaaaaaaaa          mnemonic
        002   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
                                            .
                                            .
        nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo     cccccccccc aaaaaaaaa
```

Figure 6-12. 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    oooooooooooooooooooo    cccccccccc aaaaaaaaa          mnemonic
                                        .
                                        .
      nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo    cccccccccc aaaaaaaaa

   FRAGMENT   dd    SIZE = dddd  ADDRESS = oooooo
      nnn   hhhhhhhhhhhhhhh    oooooooooooooooooooo    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 network data blocks, 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 6-12. 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 6-13 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.

```
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 6-13. NETSTC Utility FORTRAN Call Statement Format

NETLGS Utility

NETLGS calls use the same syntax and calling sequences as other AIP calls. (See sections 4 and 5). Figure 6-14 shows the NETLGS utility FORTRAN call statement format. NETLGS allows an application to enter messages into the statistical log file ZZZZZSN.

```
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 6-14. 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 6-15 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 FOR PROGRAM USE

If an application program needs to use any of the features described in Types of Application Programs earlier in this section, 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.

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.

```
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 accumulation period, expressed in hours, minutes, and seconds
d...d	Indicates decimal digits

Figure 6-15. General Format of One Period Listing in Statistical File

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 primary 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 console device connections to other programs. A passive device configured with the Remote Batch Facility as its primary 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.

This section 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 support application-to-application connections. RMV2 contains no logic requiring it to be configured as a unique identifier application program. 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.

Devices accessing RMV2 can be configured with RMV2 as an initial application program if they have a device type of console.

JOB COMMAND PORTION

Program RMV3 was run using the job commands shown in figure 7-1. The user name appearing on the NOS USER command 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 7-2 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 7-3. This figure does not illustrate login dialog or dialog after RMV2 is disconnected from the device. 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 7-4 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 7-2 to produce a processing flow diagram for the connection involved. Figure 7-5 shows the statistical file listing produced by the AIP code in RMV3.

```
RMV3. <---------------------- Job name command.
USER(APPL1,PASS,FAM1)
CHARGE(0059,2934657)
ATTACH(RMV)
FTN5(I=RMV,LO=S/-A)
LDSET(LIB=NETIOD) <---------- Uses debug and statistical file optional code version of AIP.

GET(RELJOB) <---------------- File containing NOS commands for NETREL call.

LGO.

REWIND(ZZZZZSN) )
                 |           Disposes of local files containing statistical file and debug log file
DLFP(I=0)        }           by copying the first one to OUTPUT and executing the postprocessor to
                 |           completely list the contents of the second one.
COPY(ZZZZZSN)   )
```

Figure 7-1. Command Portion of RMV3 Job

```
      PROGRAM RMV3     74/74     OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 1
   DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
   FTN5,I=RMV,L=OUTPUT,LO=S/-A.


      PROGRAM RMV3


C NAM 1 REFERENCE MANUAL SAMPLE PROGRAM
C ECHOS INTERACTIVE CONSOLE OPERATOR INPUT

C NOTE THAT THE DEBUG LOG FILE AND STATISTICAL FILE LOCAL NAMES
C ARE NOT REQUIRED ON THE PROGRAM STATEMENT GIVEN ABOVE.


      IMPLICIT INTEGER(A-Z)
      COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
      COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
      COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
      COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR

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),US1
      COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC
      EXTERNAL REPREV,CHKSUM


C INITIALIZE AND SET CONSTANTS

C SET UP LOCAL FILE NAME FOR NETREL CALLS

      DATA LFN/L"RELJOB"/

C FILE RELJOB CONTAINS THE FOLLOWING COMMANDS:

C     RELJOB.
C     USER(APPL1,PASS,FAM1)
C     CHARGE(0059,2934657)
C     DLFP(I=0)

C THIS IS THE CIRCULAR OUTPUT STACK FOR EACH CONNECTION

      DATA INSTAK, OUTSTAK/20*0,20*0/


C K IS THE APPLICATION BLOCK NUMBER COUNTER

      DATA K/20*1/


C THESE ARE NSUP WORD FIELD MASKS

      DATA S/O"02000000000000000000"/
      DATA I/O"04000000000000000000"/
      DATA MC/O"00000000007777777777"/
```

Figure 7-2. Program Portion of RMV3 (Sheet 1 of 24)

```
       PROGRAM RMV3    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 2



C THESE ARE BREAK-PROCESSING FLAGS

      DATA INTRCHR,CHANRST,CHANCLR/0,0,0/


C THIS INITIALIZES THE FLOW CONTROL ALGORITHM FOR ALL
C POSSIBLE CONNECTIONS

      DATA ABL,NB,NACN,ACN,ABHIBU,STAK/20*0,20*0,20*0,0,0,20*0/


C PACK MASK FOR CHARACTERS THAT COMPRISE OPERATOR END-CONNECTION
C COMMAND FOR NORMAL DISCONNECTION PROCESSING
C WHICH IS THE CAPITALIZED COMMAND ENDCN IN 12-BIT BYTES

      DATA ENDCN/O"01050116010401030116"/


C PACK MASK FOR CHARACTERS THAT COMPRISE OPERATOR SHUTDOWN
C COMMAND FOR NORMAL PROGRAM TERMINATION PROCESSING,
C WHICH IS THE CAPITALIZED COMMAND SHUTD IN 12-BIT BYTES

      DATA SHUTD/O"01230110012501240104"/


C PACK A CONSTANT FOR SUPERVISORY MESSAGE HEADER WORDS

      DATA SMHDR/O"03000000000004000001"/


C PACK A CONSTANT HEADER WORD FOR DISPLAY CODED OUTPUT
C OF BLOCK TYPE 2.  NOTE THAT THE NO-FORMAT-EFFECTOR BIT IS NOT SET
C BECAUSE ALL OUTPUT TO THE DEVICE GENERATED BY THE PROGRAM CONTAINS
C A FORMAT EFFECTOR CHARACTER.

      DATA DSHDR/O"02000000000020000024"/


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 PACK A CONSTANT HEADER WORD FOR DISPLAY CODED OUTPUT
C OF BLOCK TYPE 1.  NOTE THAT THE NO-FORMAT-EFFECTOR BIT IS NOT SET
C BECAUSE ALL OUTPUT TO THE DEVICE GENERATED BY THE PROGRAM CONTAINS
C A FORMAT EFFECTOR CHARACTER.

      DATA DSHDR1/O"01000000000020000024"/

C  AGAIN, ONLY 10 CHARACTERS ARE PERMITTED, PLUS A TERMINATOR WORD.


C CREATE MASK FOR UNIT SEPARATOR INSERTION CODE

      DATA US,US1/O"00370000000000000000",O"70370000000000000000"/

```

Figure 7-2. Program Portion of RMV3 (Sheet 2 of 24)

```
      C SET UP REPRIEVAL CODE TO SALVAGE DEBUG AND STATISTICAL FILES

            CALL RECOVR(REPREV,O"277",LOCF(CHKSUM))


      C SET UP ALL OTHER VARIABLES AND CONSTANTS

            CALL SETUP


      C ESTABLISH ACCESS TO THE NETWORK AND BEGIN DEBUG LOG
      C FILE CREATION

            CALL NETON("RMV2",NSUP,NSTAT,1,20)


      C TEST FOR ACCESS COMPLETION

            IF (NSTAT.NE.0) THEN
1            PRINT 100, NSTAT
1       100  FORMAT (' NSTAT = ',O20)
1            STOP 111
1           END IF
1
1
1     C UPDATE NSUP FLAGS, THEN PERFORM CONNECTION ESTABLISHMENT PROCESSING
1     C AND DISPOSE OF OTHER SUPERVISORY MESSAGES RECEIVED.
1
         15 CALL NETWAIT(4095,0)
         16 SHUTDWN=0
            SYNC=0
            CALL LOOKSM (SHUTDWN,L,SYNC)


      C RETURN FROM FC/ACK/R

         17 IF (L.EQ.1) THEN
1              GO TO 9
1
1
1     C RETURN FROM CON/REQ/R
1
1           ELSE IF (L.EQ.2) THEN
1              GO TO 15
1
1
1     C RETURN FROM FC/INIT/R
1
1           ELSE IF (L.EQ.3) THEN
1              GO TO 41
1
1
1     C RETURN FROM INTR/USR/R
1
1           ELSE IF (L.EQ.4) THEN
1              IF(INTRCHR.EQ.0) THEN
```

Figure 7-2. Program Portion of RMV3 (Sheet 3 of 24)

```
2                       GO TO 9
2                    ELSE
2                       GO TO 551
2                    END IF
2
2
2          C RETURN FROM FC/INA/R
2
1                 ELSE IF (L.EQ.5) THEN
1                    GO TO 9
1
1
1          C RETURN FROM CON/CB/R
1
1                 ELSE IF (L.EQ.6) THEN
1                    GO TO 9
1
1
1          C RETURN FROM FC/NAK/R
1
1                 ELSE IF (L.EQ.7) THEN
1                    GO TO 9
1
1
1          C RETURN FROM ERR/LGL/R
1
1                 ELSE IF (L.EQ.8) THEN
1                    GO TO 9
1
1
1          C RETURN FROM HOP/XX/R
1
1                 ELSE IF (L.EQ.9) THEN
1                    GO TO 9
1
1
1          C RETURN FROM CON/END/R
1
1                 ELSE IF (L.EQ.10) THEN
1                    GO TO 9
1
1
1          C RETURN FROM SHU/INS/R
1
1                 ELSE IF (L.EQ.11) THEN
1                    GO TO 554
1
1
1          C RETURN FROM BI/MARK/R
1
1                 ELSE IF (L.EQ.12) THEN
1                    GO TO 551
1
1
1          C RETURN FROM BAD BLOCK
1
1                 ELSE
```

Figure 7-2. Program Portion of RMV3 (Sheet 4 of 24)

```
            GO TO 777
         END IF


C INITIALIZE CONNECTION BY SENDING OUTPUT

   41 LASTBLK=1


C SEND IDENTIFYING BANNER AS FIRST OUTPUT AFTER INITIAL CONNECTION

      SEND=1
      HA=DSHDR1
      CALL NSTORE(HA,L"ABHADR",ACN)
      TA(1)="1RMV2 VER3"
      TA(2)=0
      CALL OUTPT (SEND)


C NOTE THAT ALL CONNECTIONS ARE SERVICED AS FULL-DUPLEX ON THE
C APPLICATION PROGRAM'S END

   40 CALL PROMPT (SEND)
      LASTBLK=0
   39 CALL OUTPT (SEND)
      IF (SEND .EQ. 0) GO TO 38
      IF (STAK(ACN) .EQ. 1) THEN
         SEND=0
         GO TO 39
         ELSE IF (LASTBLK.EQ.1) THEN
         GO TO 40
      ELSE
         GO TO 9
      END IF


C PAUSE TO ALLOW OUTPUT QUEUE TO CLEAR

   38 CALL NETWAIT(2,1)
      SHUTDWN=0
      SYNC=0
      CALL LOOKSM (SHUTDWN,L,SYNC)
      IF (L.EQ.1) THEN
         SEND=0
         GO TO 39
      ELSE IF (L.EQ.2) THEN
         IF(INTRCHR.EQ.0) THEN
            GO TO 9
         ELSE
            GO TO 551
         END IF
      ELSE IF (L.EQ.3) THEN
         GO TO 41
      ELSE IF (L.EQ.4) THEN
         GO TO 38
      ELSE IF (L.EQ.5) THEN
         GO TO 9
```

Figure 7-2. Program Portion of RMV3 (Sheet 5 of 24)

```
          PROGRAM RMV3    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 6

1                     ELSE IF (L.EQ.6) THEN
1                        GO TO 15
1                     ELSE IF (L.EQ.7) THEN
1                        GO TO 9
1                     ELSE IF (L.EQ.8) THEN
1                        GO TO 9
1                     ELSE IF (L.EQ.9) THEN
1                        GO TO 9
1                     ELSE IF (L.EQ.10) THEN
1                        GO TO 15
1                     ELSE IF (L.EQ.11) THEN
1                        GO TO 554
1                     ELSE IF (L.EQ.12) THEN
1                        GO TO 551
1                     ELSE
1                        GO TO 38
1                     END IF
1
1
1               C PAUSE FOR INPUT DATA OR A SUPERVISORY MESSAGE
1
                    9 CALL NETWAIT(4095,0)


                C TEST FOR QUEUED MESSAGES OR DATA BLOCKS

                  777 IF((NSUP.AND.S).NE.0) GO TO 16


                C FETCH QUEUED INPUT FROM A DEVICE

                      ALN=1
                      CALL NETGETL(ALN,HA,TA,10)


                C UNPACK THE BLOCK HEADER FOR THE DELIVERED INPUT BLOCK

                  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")
                      ABHCAN=NFETCH(HA,L"ABHCAN")
                      ABHIBU=NFETCH(HA,L"ABHIBU")
                      TLC=NFETCH(HA,L"ABHTLC")


                C BRANCH TO PROCESS DATA BLOCK OR SYNCHRONOUS SUPERVISORY MESSAGE

                      IF (ABT.EQ.3) THEN
1                        SYNC=1
1                        CALL LOOKSM (SHUTDWN,L,SYNC)
1                        GO TO 17
1                     END IF
1
1
1               C MAKE ANOTHER ATTEMPT TO FETCH QUEUED BLOCK
```

Figure 7-2. Program Portion of RMV3 (Sheet 6 of 24)

PROGRAM RMV3   74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 7

```
      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 9


C TEST FOR THROW-AWAY INPUT

      IF(ABHCAN.EQ.1) GO TO 40


C TEST FOR TYPE-IN OF ENDCN COMMAND

      IF(TA(1).EQ.ENDCN) GO TO 444


C TEST FOR TYPE-IN OF SHUTD COMMAND

      IF(TA(1).EQ.SHUTD) GO TO 666


C PROCESS ECHOABLE TEXT

      CALL PACK (SEND)
      GO TO 39


C PROCESS USER BREAKS

  551 IF((CHANCLR.EQ.1).AND.(CHANRST.EQ.1)) THEN


C TELL THE DEVICE OPERATOR WHAT HAPPENED

         IF (INTRCHR.EQ.3) TA(1)=" BREAK 1  "
         IF (INTRCHR.EQ.4) TA(1)=" BREAK 2  "
         HA=DSHDR1
         TA(2)=0
         CALL NSTORE(HA,L"ABHADR",ACN)
         LASTBLK=1
         SEND=1
         CALL OUTPT(SEND)
         CHANCLR=CHANRST=INTRCHR=0
         GO TO 40
      ELSE
         GO TO 9
      END IF


C DISCONNECT THIS TERMINAL DEVICE

  444 SMTA(1)=SMTA(2)=0
      CALL NSTORE(SMTA,L"PFCSFC",CONEND)
      CALL NSTORE(SMTA,L"RC",0)


C PASS CONNECTION DIRECTLY TO IAF WITHOUT DIALOG
```

Figure 7-2. Program Portion of RMV3 (Sheet 7 of 24)

```
       PROGRAM RMV3    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 8

      CALL NSTORE(SMTA,L"CONANM",R"IAF    ")
      SMHA=SMHDR + O"1"
      CALL NSTORE(SMTA,L"CONACN",ACN)
      NACN(ACN)=0
      CALL NETPUT(SMHA,SMTA)
      GO TO 9

  666 CALL SHUTDN

  554 STOP
      END
```

Figure 7-2. Program Portion of RMV3 (Sheet 8 of 24)

```
SUBROUTINE LOOKSM    74/74   OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.


                SUBROUTINE LOOKSM (SHUTDWN,L,SYNC)


          C PROCESS INCOMING SUPERVISORY MESSAGES

                IMPLICIT INTEGER (A-Z)
                COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
                COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
                COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
                COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
                COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
                COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC


          C PROCESS SYNCHRONOUS SUPERVISORY MESSAGES

                IF (SYNC.EQ.1) THEN
1                  SMHA=HA
1                  DO 2 I=1,63
1                  SMTA(I)=TA(I)
1             2    CONTINUE
1                  GO TO 1
1
1               ELSE
1                  GO TO 3
1
1               END IF
1
1
1         C WAIT FOR AN ASYNCHRONOUS SUPERVISORY MESSAGE IF NECESSARY
1
              3 IF ((NSUP.AND.S).EQ.0) THEN
1                  IF(((NSUP.AND.I).EQ.0).AND.(SHUTDWN.EQ.0)) THEN
2                     CALL NETWAIT(4095,0)
2
2         C RETURN TO FETCH INPUT DATA
2
2                     RETURN
2
2                  ELSE
2                     L=13
2                     RETURN
2
2                  END IF
1               END IF
1
1
1         C FETCH AN ASYNCHRONOUS SUPERVISORY MESSAGE FROM ACN=0
1         C ON LIST ZERO
1
                ALN=0
                CALL NETGETL(ALN,SMHA,SMTA,63)


          C UNPACK THE MESSAGE IDENTIFICATION AND BRANCH ON THE TYPE
```

Figure 7-2. Program Portion of RMV3 (Sheet 9 of 24)

```
           1 PFCSFC=NFETCH(SMTA,L"PFCSFC")
             PFC=NFETCH(SMTA,L"PFC")


       C NOTE THAT THIS CODE EXITS WITH THE L VALUE SET SO THAT IT CAN BE
       C USED FOR BRANCHING IN THE MAIN PROGRAM ON RETURN FROM LOOKSM

             IF (PFCSFC.EQ.SM(1)) THEN
1               L=1
1               GO TO 10
1            ELSE IF (PFCSFC.EQ.SM(2)) THEN
1               L=2
1               GO TO 20
1            ELSE IF (PFCSFC.EQ.SM(3)) THEN
1               L=3
1               GO TO 30
1            ELSE IF (PFCSFC.EQ.SM(4)) THEN
1               L=4
1               GO TO 50
1            ELSE IF (PFCSFC.EQ.SM(5)) THEN
1               L=5
1               GO TO 60
1            ELSE IF (PFCSFC.EQ.SM(6)) THEN
1               L=6
1               GO TO 70
1            ELSE IF (PFCSFC.EQ.SM(7)) THEN
1               L=7
1               GO TO 80
1            ELSE IF (PFCSFC.EQ.SM(8)) THEN
1               L=8
1               GO TO 90
1            ELSE IF (PFCSFC.EQ.SM(9)) THEN
1               L=9
1               DO 9 M=1,7
1               IF(PFCSFC.EQ.SSM(M))GOTO(11,21,31,41,51,61,71),M
1          9    CONTINUE
1            ELSE IF (PFCSFC.EQ.SM(10)) THEN
1               L=10
1               GO TO 110
1            ELSE IF (PFCSFC.EQ.SM(11)) THEN
1               L=11
1               GO TO 120
1            ELSE IF (PFCSFC.EQ.SM(12)) THEN
1               L=12
1               GO TO 130
1
1
1      C TEST FOR END OF MESSAGE BRANCHING TABLE
1
1            ELSE
1               L=13
1            END IF
1
1
1      C PROCESS UNRECOGNIZED SUPERVISORY MESSAGE CODE
1
```

Figure 7-2. Program Portion of RMV3 (Sheet 10 of 24)

```
      SUBROUTINE LOOKSM    74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 3

                   IF (SM(L).EQ.999) THEN

             C ISSUE DIAGNOSTIC MESSAGE TO OUTPUT FILE

1                   PRINT 1000, SMHA,SMTA
1             1000  FORMAT (' COULD NOT FIND SM IN TABLE OF SUPPORTED CODES',
1                  * //' HA = ',O20,/' TA = ',/63(1X,O20/))
1
1                  END IF
1
1
1            C TRY AGAIN
1
                   GO TO 3


             C PROCESS FC/ACK/R SUPERVISORY MESSAGE

                10 ACN=NFETCH(SMTA,L"FCACN")
                   IABN(ACN)=NFETCH(SMTA,L"FCABN")

             C UPDATE FLOW CONTROL ALGORITHM

                   NB(ACN)=NB(ACN) - 1
                   RETURN


             C PROCESS CON/REQ/R SUPERVISORY MESSAGE

             C UNPACK MESSAGE AND USE CONTENTS TO SET UP CONNECTION
             C FLOW CONTROL ALGORITHM

                20 ACN=NFETCH(SMTA,L"CONACN")
                   ABL(ACN)=NFETCH(SMTA,L"CONABL")
                   DT=NFETCH(SMTA,L"CONDT")
                   NB(ACN)=0

             C PACK CON/REQ/N OR CON/REQ/A MESSAGE

                   SMTA(1)=0
                   CALL NSTORE(SMTA,L"PFCSFC",L"CONREQ")
                   CALL NSTORE(SMTA,L"CONACN",ACN)

             C SET RESPONSE BIT TO ACCEPT OR REJECT CONNECTION

                   IF (DT.EQ.0) CALL NSTORE (SMTA,L"RB",1)
                   IF (DT.NE.0) CALL NSTORE (SMTA,L"EB",1)

             C INPUT MUST BE ASCII IN 12-BIT BYTES

                   CALL NSTORE(SMTA,L"CONACT",3)

             C ASSIGN ALL INTERACTIVE CONSOLES TO LIST 1

                   CALL NSTORE(SMTA,L"CONALN",1)
                   SMHA=SMHDR
```

Figure 7-2. Program Portion of RMV3 (Sheet 11 of 24)

```
      SUBROUTINE LOOKSM   74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 4

C SEND THE CONNECTION-ACCEPTED OR CONNECTION-REJECTED SUPERVISORY MESSAGE

      CALL NETPUT(SMHA,SMTA)

      RETURN


C PROCESS FC/INIT/R SUPERVISORY MESSAGE

C SET THE RESPONSE BIT TO INDICATE READY FOR
C TRANSMISSION TO BEGIN

   30 CALL NSTORE(SMTA,L"RB",1)

C DETERMINE LOGICAL CONNECTION INVOLVED AND UPDATE
C CONNECTION TABLE

      ACN=NFETCH(SMTA,L"FCACN")
      NACN(ACN)=1
      SMHA=SMHDR
      IABN(ACN)=ABN(ACN)=0

C SEND THE CONNECTION-INITIALIZED MESSAGE

      CALL NETPUT(SMHA,SMTA)

      RETURN


C PROCESS INTR/USR/R SUPERVISORY MESSAGE

   50 ACN=NFETCH(SMTA,L"INTRACN")
      INTRCHR=NFETCH(SMTA,L"INTRCHR")

C PACK RESPONSE MESSAGE AND CLEAR FLOW CONTROL PARAMETERS

      SMTA(1)=0
      SMHA=SMHDR
      CALL NSTORE (SMTA,L"PFCSFC",INTRRSP)
      CALL NSTORE (SMTA,L"INTRACN",ACN)
      CALL NETPUT (SMHA,SMTA)

C IF THIS IS A USER BREAK, CLEAR THE OUTPUT QUEUE

      IF ((INTRCHR.EQ.3).OR.(INTRCHR.EQ.4)) THEN
         CHANRST=1
         INSTAK(ACN)=OUTSTAK(ACN)=STAK(ACN)=0

      END IF


C TELL THE DEVICE OPERATOR WHAT HAPPENED

      IF ((INTRCHR.NE.3).AND.(INTRCHR.NE.4)) THEN
         TA(1)=" BYPASSED "
         HA=DSHDR1
         TA(2)=0
```

Figure 7-2. Program Portion of RMV3 (Sheet 12 of 24)

```
      SUBROUTINE LOOKSM    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 5

               CALL NSTORE(HA,L"ABHADR",ACN)
               SEND=1
               LASTBLK=1
               CALL OUTPT (SEND)
               CALL PROMPT(SEND)
               LASTBLK=0
               CALL OUTPT(SEND)
               INTRCHR=0

               RETURN

            END IF
            RETURN


C PROCESS FC/INACT/R SUPERVISORY MESSAGE

C UPDATE CONNECTION TABLE

   60 ACN=NFETCH(SMTA,L"FCACN")
      NACN(ACN) = 0
      HA=DSHDR
      CALL NSTORE(HA,L"ABHADR",ACN)


C OUTPUT DISCONNECTION INDICATOR TO POSSIBLE OPERATOR

      TA(1)=" TIME OUT "
      TA(2)=0


C NOTE THAT RMV2 DOES NOT WAIT FOR AN FC/ACK/R CORRESPONDING TO
C THIS OUTPUT MESSAGE. AN ERR/LGL/R MESSAGE WILL EVENTUALLY
C BE CAUSED BY THE CONNECTION TERMINATION PROCESSING CODE,
C CAUSING RMV2 TO NETOFF WITHOUT DEVICE OPERATOR
C OR HOST OPERATOR ACTION BEING REQUIRED.

      INSTAK(ACN)=OUTSTAK(ACN)=STAK(ACN)=0
      SEND=1
      LASTBLK=0
      CALL OUTPT (SEND)


C PACK AND SEND CONNECTION-END REQUEST MESSAGE

      SMTA(1)=0
      CALL NSTORE(SMTA,L"PFCSFC",CONEND)
      CALL NSTORE(SMTA,L"CONACN",ACN)
      SMTA(2)=0
      SMHA=SMHDR
      CALL NETPUT (SMHA,SMTA)
      RETURN


C PROCESS CON/CB/R SUPERVISORY MESSAGE

   70 ACN=NFETCH(SMTA,L"CONACN")
```

Figure 7-2. Program Portion of RMV3 (Sheet 13 of 24)

```
      SUBROUTINE LOOKSM   74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 6

            PRINT 75,ACN
         75 FORMAT(' CONNECTION BROKEN, ACN = ',I3)


      C FETCH ALL OUTSTANDING INPUT BLOCKS UNTIL A NULL
      C BLOCK IS RECEIVED

         73 CALL NETGET(ACN,HA,TA,63)
            IF (NFETCH(HA,L"ABHABT").EQ.0) GO TO 72


      C     DETERMINE WHETHER THIS IS A NORMAL SHUTD SEQUENCE FETCHED OUT OF
      C     SYNCHRONIZATION.  IF SO, USE THE ERR/LGL/R LOGIC TO SHUT DOWN.

            IF(TA(1).EQ.SHUTD) GO TO 76
            GO TO 73


      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 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 PROCESS CON/END/N SUPERVISORY MESSAGE
      C PROCESSING TREATS THE MESSAGE AS ADVISORY IN ALL CASES.

        110 MSGLTH=410
            NREWIND=0
            IF((NSUP.AND.MC).GT.255) CALL NETREL(LFN,MSGLTH,NREWIND)

            RETURN


      C PROCESS ERR/LGL/R SUPERVISORY MESSAGE,
      C WRITE MESSAGE TO OUTPUT FILE FOR ANALYSIS, THEN SHUT
      C DOWN OPERATIONS

         90 PRINT 1001,SMHA,SMTA
       1001 FORMAT(1X,"HA = ",O20,/1X,"TA = ",/1X,O20,1X,O20/,1X,O20)
```

Figure 7-2. Program Portion of RMV3 (Sheet 14 of 24)

```
      SUBROUTINE LOOKSM    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 7

   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 UPDATE CONNECTION TABLE

         NACN(II)=0
      END IF

  333 CONTINUE

      CALL NETOFF
      STOP 247


C PROCESS HOST OPERATOR TURN-DEBUGGING-ON COMMAND

   11 CONTINUE
      RETURN


C PROCESS HOST OPERATOR TURN-DEBUGGING-OFF COMMAND

   21 CONTINUE
      RETURN


C PROCESS HOST OPERATOR DUMP-FIELD-LENGTH COMMAND

   31 DUMPID=1
      ECS=1
      CALL NETDMB (DUMPID,ECS)

      RETURN


C PROCESS HOST OPERATOR STOP-LOGGING COMMAND

   41 DBUGSUP=1
      DUBDAT=1
      CALL NETDBG (DBUGSUP,DBUGDAT,AVAIL)

      RETURN


C PROCESS HOST OPERATOR START-LOGGING COMMAND

   51 DBUGSUP=0
      DBUGDAT=0
      CALL NETDBG (DBUGSUP,DBUGDAT,AVAIL)

```

Figure 7-2. Program Portion of RMV3 (Sheet 15 of 24)

```
        SUBROUTINE LOOKSM    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 8

                  RETURN


        C PROCESS HOST OPERATOR RELEASE-LOG-FILE COMMAND

             61 MSGLTH=410
                NREWIND=0
                CALL NETREL (LFN,MSGLTH,NREWIND)

                RETURN


        C PROCESS HOST OPERATOR RESTART-STATISTICS COMMAND

             71 ONOFF=0
                CALL NETSTC (ONOFF,AVAIL)

                RETURN


        C PROCESS THE BIMARK SYNCHRONOUS SUPERVISORY MESSAGE

            130 HA=SMHDR
                TA(1)=0
                CALL NSTORE (HA,L"ABHADR",ACN)
                CALL NSTORE(HA,L"ABHACT",2)
                CALL NSTORE(HA,L"ABHTLC",2)
                CALL NSTORE(TA(1),L"PFCSFC",ROMARK)
                CALL NETPUT(HA,TA(1))
                CHANCLR=1

                RETURN


        C PROCESS SHUT/INSD/R SUPERVISORY MESSAGE, THEN
        C SHUTDOWN OPERATIONS

        C DETERMINE TYPE OF SHUTDOWN

            120 IBIT=NFETCH(SMTA,L"SHUTF")


        C IF THIS IS A FORCED SHUTDOWN, STOP NOW
                IF (IBIT.EQ.1) THEN
1                  CALL NETOFF
1                  STOP 313
1
1               END IF
1
1
1       C SHUTDOWN GRACEFULLY IF TIME PERMITS BY
1       C DISCONNECTING ALL TERMINAL DEVICES
1
                CALL SHUTDN
                END
```

Figure 7-2. Program Portion of RMV3 (Sheet 16 of 24)

```
SUBROUTINE OUTPT    74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 1
 DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
 FTN5,I=RMV,L=OUTPUT,LO=S/-A.

              SUBROUTINE OUTPT (SEND)


        C OUTPUT ONE DATA BLOCK

              IMPLICIT INTEGER (A-Z)
              COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
              COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
              COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
              COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
              COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
              COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC


        C IS THERE DATA IN THE MAIN OUTPUT BUFFER?

              IF (SEND.EQ.1) THEN

        C        IF SO, IS THERE SOMETHING ELSE TO SEND FIRST?

1               IF (STAK(ACN) .EQ. 1) THEN
1
1       C           IF SO, ADD NEW OUTPUT TO STACK
1
2                   GO TO 1
2               ELSE
2
2       C        IF NOT, TEST IF NEW OUTPUT CAN BE SENT
2
2                   GO TO 9
2               END IF
1             ELSE
1
1
1       C IF NOT, TEST IF DATA NEEDS TO BE SENT FROM THE STACK
1
1               GO TO 8
1             END IF
1
1       C IS THERE DATA IN THE STACK?
1
            8 IF (STAK(ACN) .EQ. 0) THEN

        C        IF NOT, EXIT

1               RETURN
1             ELSE
1
1       C        IF SO, TEST IF IT CAN BE SENT
1
1               GO TO 3
1             END IF
1
1
1       C CAN DATA BE SENT?
```

Figure 7-2. Program Portion of RMV3 (Sheet 17 of 24)

```
      SUBROUTINE OUTPT    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 2


    9 IF (((NB(ACN).GE.ABL(ACN)).AND.(CHANCLR.EQ.0)).AND.
     +    (CHANRST.EQ.0)) THEN

C         IF NOT, STACK IT

          STAK(ACN)=OUTSTAK(ACN)=INSTAK(ACN)=1
          OVRFLHA(INSTAK(ACN),ACN)=HA
          DO 888 JJ=1, 63, 1
  888     OVRFLTA(JJ,INSTAK(ACN),ACN)=TA(JJ)
          RETURN

C         IF SO, DO IT

      ELSE

C         UPDATE FLOW CONTROL ALGORITHM

          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
      END IF


C IS THERE ROOM FOR MORE DATA IN THE STACK?

C     IF NOT, THROW AWAY NEW OUTPUT

    1 IF (INSTAK(ACN).GT.OUTSTAK(ACN)) THEN
          IF ((INSTAK(ACN) - OUTSTAK(ACN)) .EQ. 7) THEN
              SEND=0
              RETURN
          END IF
      ELSE
          IF ((OUTSTAK(ACN) - INSTAK(ACN)) .EQ. 1) THEN
              SEND=0
              RETURN
          END IF
      END IF
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 PROCESS DATA ALREADY IN STACK

C CAN DATA BE SENT?

    3 IF (NB(ACN) .GE. ABL(ACN)) THEN
```

Figure 7-2. Program Portion of RMV3 (Sheet 18 of 24)

```
              C       IF NOT, EXIT

1                     RETURN
1
1             C       IF SO, DO IT
1
1                   ELSE
1
1             C       UPDATE FLOW CONTROL ALGORITHM
1
1                     ABN(ACN)=ACN*64 + K(ACN)
1                     K(ACN)=K(ACN) + 1
1                     NB(ACN)=NB(ACN) + 1
1                     CALL NSTORE(OVRFLHA(OUTSTAK(ACN),ACN),L"ABHABN",ABN(ACN))
1                     CALL NETPUT(OVRFLHA(OUTSTAK(ACN),ACN),
1                  +     OVRFLTA(1,OUTSTAK(ACN),ACN))
1
1             C       TEST IF STACK HAS BEEN EMPTIED
1
1                     IF (OUTSTAK(ACN).EQ.INSTAK(ACN)) THEN
2                         STAK(ACN)=0
2
2             C           IF SO, REINITIALIZE POINTERS
2
2                         OUTSTAK(ACN)=INSTAK(ACN)=0
2                     ELSE
2
2             C           IF NOT, MOVE THE SEND BUFFER POINTER FOR NEXT PASS
2
2                         OUTSTAK(ACN)=OUTSTAK(ACN) + 1
2                         IF (OUTSTAK(ACN) .EQ. 9) OUTSTAK(ACN)=1
2                         RETURN
2                     END IF
1                   END IF
1
                    RETURN
                    END
```

Figure 7-2. Program Portion of RMV3 (Sheet 19 of 24)

```
SUBROUTINE PROMPT   74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.


                SUBROUTINE PROMPT (SEND)

                IMPLICIT INTEGER (A-Z)
                COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
                COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
                COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
                COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
                COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
                COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC

                HA=DSHDR
                CALL NSTORE(HA,L"ABHADR",ACN)
                TA(1)=" INPUT PLS"
                TA(2)=0
                SEND=1
                RETURN
                END
```

Figure 7-2. Program Portion of RMV3 (Sheet 20 of 24)

```
SUBROUTINE SETUP    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.


      SUBROUTINE SETUP

      IMPLICIT INTEGER(A-Z)
      COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
      COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
      COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
      COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
      COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
      COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC


C SET OUTGOING SUPERVISORY MESSAGE CONSTANTS

      CONEND=NFETCH(0,L"CONEND")
      ROMARK=NFETCH(0,L"ROMARK")
      INTRRSP=NFETCH(0,L"INTRRSP")


C BUILD A BRANCHING TABLE FOR INCOMING 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"INTRUSR")
      SM(5)=NFETCH(0,L"FCINA")
      SM(6)=NFETCH(0,L"CONCB")
      SM(7)=NFETCH(0,L"FCNAK")
      SM(8)=NFETCH(0,L"ERRLGL")
      SM(9)=NFETCH(0,L"HOP")
      SM(10)=NFETCH(0,L"CONEND")


C     SET RESPONSE BIT FOR THE CON/END/N MESSAGE

      SM(10)=SM(10) .OR.0"100"
      SM(11)=NFETCH(0,L"SHUINS")
      SM(12)=NFETCH(0,L"BIMARK")
      SM(13)=999


C BUILD A BRANCHING TABLE FOR HOST OPERATOR COMMANDS

      SSM(1)=NFETCH(0,L"HOPDB")
      SSM(2)=NFETCH(0,L"HOPDE")
      SSM(3)=NFETCH(0,L"HOPDU")
      SSM(4)=NFETCH(0,L"HOPNOTR")
      SSM(5)=NFETCH(0,L"HOPTRCE")
      SSM(6)=NFETCH(0,L"HOPREL")
      SSM(7)=NFETCH(0,L"HOPRS")

      RETURN
      END
```

Figure 7-2. Program Portion of RMV3 (Sheet 21 of 24)

```
SUBROUTINE PACK     74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17     PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.


            SUBROUTINE PACK (SEND)

            IMPLICIT INTEGER(A-Z)
            COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
            COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
            COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
            COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
            COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
            COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC


      C CREATE HEADER WORD TO ECHO INPUT AS OUTPUT

            HA =(HA .AND. O"77777777777774007777") + O"1"


      C CHANGE APPLICATION BLOCK TYPE TO 1
            IF (ABT.EQ.2) CALL NSTORE (HA,L"ABHABT",1)
            IF (ABT.EQ.2) THEN
1              LASTBLK=1
1           ELSE
1              LASTBLK=0
1           END IF
1
1
1     C INHIBIT FIRST CHARACTER AS A FORMAT EFFECTOR
1
            CALL NSTORE(HA,L"ABHNFE",1)


      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
1            TA(FWP1)=US
1           ELSE
1            XXX=SHIFT(US1,-XTRA)
1            YYY=SHIFT(US,-XTRA)
1
1
1     C ZERO OUT REMAINDER OF WORD AND ADD UNIT SEPARATOR  CHARACTER TO END OF BLOCK
1
1            TA(FWP1)=TA(FWP1) .AND. XXX .OR. YYY
1           END IF
1
            SEND=1
            RETURN
            END
```

Figure 7-2. Program Portion of RMV3 (Sheet 22 of 24)

```
SUBROUTINE SHUTDN    74/74    OPT=0,ROUND= A/ S/ M/-D,-DS    FTN 5.1+599    83/08/05. 11.38.17    PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.


            SUBROUTINE SHUTDN

            IMPLICIT INTEGER(A-Z)
            COMMON /RMCOM/K(20),LASTBLK,I,S,NSUP,SMHDR,DSHDR,DSHDR1,NACN(20)
            COMMON /RMCOM/CONEND,ROMARK,ACN,ABN(20),SM(20),ABL(20),ABHIBU,US
            COMMON /RMCOM/NB(20),HA,INSTAK(20),OUTSTAK(20),ENDCN,SHUTD,INTRRSP
            COMMON /RMCOM/INTRCHR,CHANRST,CHANCLR
            COMMON /RMCOM/TA(63),STAK(20),OVRFLHA(8,20),OVRFLTA(63,8,20),US1
            COMMON /RMCOM/IABN(20),SMHA,SMTA(63),SSM(8),MC,LFN,ABT,ACT,TLC


      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)


      C PASS CONNECTION DIRECTLY TO IAF WITHOUT DIALOG

            CALL NSTORE(SMTA,L"CONANM",R"IAF    ")
            SMHA=SMHDR + O"1"
            DO 555 J=1,20
            IF (NACN(J).EQ.1) THEN
1            CALL NSTORE (SMTA,L"CONACN",J)
1            NACN(J)=0
1            CALL NETPUT (SMHA,SMTA)
1           END IF
        555 CONTINUE


      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
            SYNC=0
            CALL LOOKSM (SHUTDWN,L,SYNC)
            IF (L.EQ.3) GO TO 666
            IF (L.LE.12) GO TO 97


      C FINISH WRITING DEBUG LOG AND STATISTICAL FILES

            CALL NETOFF

            STOP 333
            END
```

Figure 7-2. Program Portion of RMV3 (Sheet 23 of 24)

```
SUBROUTINE REPREV   74/74   OPT=0,ROUND= A/ S/ M/-D,-DS   FTN 5.1+599   83/08/05. 11.38.17   PAGE 1
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=5000
FTN5,I=RMV,L=OUTPUT,LO=S/-A.

                SUBROUTINE REPREV (IXCHNG,IFLAG,IFLDLN)


          C THIS SUBROUTINE SALVAGES THE DEBUG AND STATISTICAL FILE ENTRIES BY
          C CALLING THE AIP ROUTINE NETOFF TO FLUSH BUFFERS IN CASE THE
          C APPLICATION PROGRAM IS ABORTED DURING EXECUTION


                DIMENSION IXCHNG(17),IFLDLN(O"50000")
                IFLAG=1

                CALL NETOFF
                STOP 10

                ENTRY CHKSUM
                END
```

Figure 7-2. Program Portion of RMV3 (Sheet 24 of 24)

RMV2 VER3	
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 host.
INPUT PLS	RMV2 prompt for next input.
SHUTD	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 next input.
ENDCN	Terminal operator entry, causes normal connection termination for this terminal. Next terminal operator dialog is with IAF, if that program is available.
INPUT PLS	RMV2 prompt for input.
Any characters other than SHUTD or ENDCN, up to 314	Terminal operator entry.
Any characters other than SHUTD or ENDCN, up to 314	RMV2 echoed output, single-spaced.
INPUT PLS	RMV2 prompt for next entry.

Figure 7-3. Possible Dialogs Supported by Sample FORTRAN Program

```
11.38.26.000         NETON    (024677)     ANAME = RMV2          DATE = 83/08/05                MSG NO. 000001
 NSUP ADDR = 000140  MINACN =00001  MAXACN =00020


11.38.53.498         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063        MSG NO. 000002
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0010

     001 630000001600200 30600000000130001000  CONREQ      C
     002 51C75F0ADB45018 24343537025555050030  T124B  E X   UP-4P
     003 000000000006EA 00000000000000003352         0)        N
     004 0000000002DD40B 00000000000013352013       K2PK        -T
     005 xxxxxxx6DB40011 xxxxxxxxxxx555000021  xxxxx    Q  M B [@
     006 xxxxxxxE1880037 xxxxxxxxxxxxxx000067  xxxxxxx  &    16A  7
     007 000FF8FFFFFFFFF 00007770777777777777    ;';;;;;;    X___
     008 FFF3400001FFFFF 77771500000007777777  ;;M   G;;;  __4___
     009 000000000000F6F 00000000000000007557            .  ‾‾   V
     010 7C014034460D1C1 37000500150430150701  4 E MDXMGA  W@  D@QA

11.38.53.508         NETPUT   (031655)   HA =024544    TA =024545                            MSG NO. 000003
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 634000001000 0C1 30640000000100000301  CONREQN      C@

11.38.54.007         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063        MSG NO. 000004
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 830700001000000 40603400000100000000  FCINIT

11.38.54.010         NETPUT   (031655)   HA =024544    TA =024545                            MSG NO. 000005
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 834700001000000 40643400000100000000  FCINITN       G

11.38.54.011         NETPUT   (031655)   HA =000315    TA =000374                            MSG NO. 000006
 ABT =01  ADR =0001  ABN =000065  ACT =04  STATUS = 00000000  TLC = 0020

     001 71235676D58549E 34221526355526052236  1RMV2 VER3  Q#VVU I
     002 000000000000000 00000000000000000000              @

11.38.54.011         NETPUT   (031655)   HA =000315    TA =000374                            MSG NO. 000007
 ABT =02  ADR =0001  ABN =000066  ACT =04  STATUS = 00000000  TLC = 0020

     001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0

11.38.54.505         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063        MSG NO. 000008
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 1 of 13)

```
     001 830200001001040 40601000000100010100  FCACK

11.38.54.509         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000009
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001001080 40601000000100010200  FCACK

11.39.10.797         NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010      MSG NO. 000010
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0047

     001 05406806502006E 01240150014500400156  ATA/A+ 5A,  THE N
     002 065078074020063 01450170016400400143  A+A'A" 5A8  EXT C
     003 068061072061063 01500141016201410143  A/A6A]A6A8  HARAC
     004 074065072020069 01640145016200400151  A"A+A] 5A(  TER I
     005 073020061020075 01630040014100400165  A% 5A6 5A   S A U
     006 07306507202D062 01630145016200550142  A%A+A]  A7  SER-B
     007 07206506106B02D 01620145014101530055  A]A+A6A$    REAK-
     008 031020063068061 00610040014301500141   [ 5A8A/A6  1 CHA
     009 072061063074065 01620141014301640145  A]A6A8A"A+  RACTE
     010 07202E000000000 01620056000000000000  A] ,        R.

11.39.10.804         NETPUT   (031655)   HA =000315   TA =000374                           MSG NO. 000011
 ABT =01  ADR =0001  ABN =000067  ACT =03  STATUS = 00001000   TLC = 0048

     001 05406806502006E 01240150014500400156  ATA/A+ 5A,  THE N
     002 065078074020063 01450170016400400143  A+A'A" 5A8  EXT C
     003 068061072061063 01500141016201410143  A/A6A]A6A8  HARAC
     004 074065072020069 01640145016200400151  A"A+A] 5A(  TER I
     005 073020061020075 01630040014100400165  A% 5A6 5A   S A U
     006 07306507202D062 01630145016200550142  A%A+A]  A7  SER-B
     007 07206506106B02D 01620145014101530055  A]A+A6A$    REAK-
     008 031020063068061 00610040014301500141   [ 5A8A/A6  1 CHA
     009 072061063074065 01620141014301640145  A]A6A8A"A+  RACTE
     010 07202E01F000000 01620056003700000000  A] , 4      R.

11.39.10.805         NETPUT   (031655)   HA =000315   TA =000374                           MSG NO. 000012
 ABT =02  ADR =0001  ABN =000068  ACT =04  STATUS = 00000000   TLC = 0020

     001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0

11.39.11.844         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000013
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 8302000010010C0 40601000000100010300  FCACK

11.39.11.850         NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000014
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 2 of 13)

```
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

    001 830200001001100 40601000000100010400  FCACK

11.39.15.953        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000015
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

    001 800003001000000 40000003000100000000  INTRUSR

11.39.15.957        NETPUT  (031655)   HA =024544   TA =024545                            MSG NO. 000016
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

    001 800100001000000 40000400000100000000  INTRRSP

11.39.16.011        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010      MSG NO. 000017
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

    001 CA0000000000000 62400000000000000000  BIMARK      J

11.39.16.043        NETPUT  (031655)   HA =000315   TA =000374                            MSG NO. 000018
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

    001 CB0000000000000 62600000000000000000  ROMARK      K

11.39.16.043        NETPUT  (031655)   HA =000315   TA =000374                            MSG NO. 000019
 ABT =01  ADR =0001  ABN =000069  ACT =04  STATUS = 00000000   TLC = 0020

    001 B4248504BB5CB6D 55022205011355345555   BREAK 1    4$  ; 6
    002 000000000000000 00000000000000000000              P

11.39.16.043        NETPUT  (031655)   HA =000315   TA =000374                            MSG NO. 000020
 ABT =02  ADR =0001  ABN =000070  ACT =04  STATUS = 00000000   TLC = 0020

    001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
    002 000000000000000 00000000000000000000              0

11.39.17.006        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000021
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

    001 830200001000000 40601000000100000000  FCACK

11.39.17.010        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000022
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

    001 830200001001140 40601000000100010500  FCACK
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 3 of 13)

```
11.39.17.014        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000023
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 830200001001180 40601000000100010600  FCACK

11.39.32.490        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010      MSG NO. 000024
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000  TLC = 0047

    001 05406806502006E 01240150014500400156  ATA/A+ 5A,   THE N
    002 065078074020063 01450170016400400143  A+A'A" 5A8   EXT C
    003 068061072061063 01500141016201410143  A/A6A]A6A8   HARAC
    004 074065072020069 01640145016200400151  A"A+A] 5A(   TER I
    005 073020061020075 01630040014100400165  A% 5A6 5A    S A U
    006 07306507202D062 01630145016200550142  A%A+A]  A7   SER-B
    007 07206506106B02D 01620145014101530055  A]A+A6A$     REAK-
    008 032020063068061 00620040014301500141   ] 5A8A/A6   2 CHA
    009 072061063074065 01620141014301640145  A]A6A8A"A+   RACTE
    010 07202E000000000 01620056000000000000  A] ,         R.

11.39.32.502        NETPUT  (031655)  HA =000315  TA =000374                             MSG NO. 000025
 ABT =01  ADR =0001  ABN =000071  ACT =03  STATUS = 00001000  TLC = 0048

    001 05406806502006E 01240150014500400156  ATA/A+ 5A,   THE N
    002 065078074020063 01450170016400400143  A+A'A" 5A8   EXT C
    003 068061072061063 01500141016201410143  A/A6A]A6A8   HARAC
    004 074065072020069 01640145016200400151  A"A+A] 5A(   TER I
    005 073020061020075 01630040014100400165  A% 5A6 5A    S A U
    006 07306507202D062 01630145016200550142  A%A+A]  A7   SER-B
    007 07206506106B02D 01620145014101530055  A]A+A6A$     REAK-
    008 032020063068061 00620040014301500141   ] 5A8A/A6   2 CHA
    009 072061063074065 01620141014301640145  A]A6A8A"A+   RACTE
    010 07202E01F000000 01620056003700000000  A] , 4       R.

11.39.32.502        NETPUT  (031655)  HA =000315  TA =000374                             MSG NO. 000026
 ABT =02  ADR =0001  ABN =000072  ACT =04  STATUS = 00000000  TLC = 0020

    001 B49390554B50313 55111620252455201423  INPUT PLS  4  UKP1
    002 000000000000000 00000000000000000000             0

11.39.34.047        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000027
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 8302000010011C0 40601000000100010700  FCACK

11.39.34.067        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000028
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 830200001001200 40601000000100011000  FCACK
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 4 of 13)

```
11.39.36.687        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000029
 ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

      001 800004001000000 40000004000100000000  INTRUSR

11.39.36.740        NETPUT  (031655)   HA =024544   TA =024545                             MSG NO. 000030
 ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

      001 800100001000000 40000400000100000000  INTRRSP

11.39.36.811        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010       MSG NO. 000031
 ABT =03   ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

      001 CA00000901DE000 62400000022007360000  BIMARK      J     ^

11.39.36.822        NETPUT  (031655)   HA =000315   TA =000374                             MSG NO. 000032
 ABT =03   ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

      001 CB0000000000000 62600000000000000000  ROMARK      K

11.39.36.822        NETPUT  (031655)   HA =000315   TA =000374                             MSG NO. 000033
 ABT =01   ADR =0001  ABN =000073  ACT =04  STATUS = 00000000   TLC = 0020

      001 B4248504BB5DB6D 55022205011355355555   BREAK 2    4$  ;]6
      002 000000000000000 00000000000000000000              P

11.39.36.823        NETPUT  (031655)   HA =000315   TA =000374                             MSG NO. 000034
 ABT =02   ADR =0001  ABN =000074  ACT =04  STATUS = 00000000   TLC = 0020

      001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
      002 000000000000000 00000000000000000000              0

11.39.37.707        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000035
 ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

      001 830200001000000 40601000000100000000  FCACK

11.39.37.711        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000036
 ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

      001 830200001001240 40601000000100011100  FCACK             $

11.39.37.715        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000037
 ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 5 of 13)

```
     001 830200001001280 40601000000100011200  FCACK            (

11.39.51.219        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010      MSG NO. 000038
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000  TLC = 0036

     001 05406806502006E 01240150014500400156  ATA/A+ 5A,  THE N
     002 065078074020065 01450170016400400145  A+A'A" 5A+  EXT E
     003 06E074072079020 01560164016201710040  A,A"A]A? 5  NTRY
     004 069073020061020 01510163004001410040  A(A% 5A6 5  IS A
     005 06207206506106B 01420162014501410153  A7A]A+A6A$  BREAK
     006 02006306F06E064 00400143015701560144   5A8A.A,A9   COND
     007 06907406906F06E 01510164015101570156  A(A"A(A.A,  ITION
     008 02E000000000000 00560000000000000000   ,          .

11.39.51.225        NETPUT  (031655)   HA =000315   TA =000374                           MSG NO. 000039
 ABT =01  ADR =0001  ABN =000075  ACT =03  STATUS = 00001000  TLC = 0037

     001 05406806502006E 01240150014500400156  ATA/A+ 5A,  THE N
     002 065078074020065 01450170016400400145  A+A'A" 5A+  EXT E
     003 06E074072079020 01560164016201710040  A,A"A]A? 5  NTRY
     004 069073020061020 01510163004001410040  A(A% 5A6 5  IS A
     005 06207206506106B 01420162014501410153  A7A]A+A6A$  BREAK
     006 02006306F06E064 00400143015701560144   5A8A.A,A9   COND
     007 06907406906F06E 01510164015101570156  A(A"A(A.A,  ITION
     008 02E01F000000000 00560037000000000000   , 4        .

11.39.51.225        NETPUT  (031655)   HA =000315   TA =000374                           MSG NO. 000040
 ABT =02  ADR =0001  ABN =000076  ACT =04  STATUS = 00000000  TLC = 0020

     001 849390554850313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0

11.39.51.747        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000041
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 8302000010012C0 40601000000100011300  FCACK            ,

11.39.51.751        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000042
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 830200001001300 40601000000100011400  FCACK            0

11.39.56.410        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000043
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

     001 800003001000000 40000003000100000000  INTRUSR
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 6 of 13)

```
11.39.56.414        NETPUT  (031655)  HA =024544  TA =024545                               MSG NO. 000044
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 8001000001000000 40000400000100000000  INTRRSP

11.39.56.464        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010       MSG NO. 000045
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

     001 CA00000000000000 62400000000000000000  BIMARK      J

11.39.56.478        NETPUT  (031655)  HA =000315  TA =000374                               MSG NO. 000046
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002

     001 CB00000000000000 62600000000000000000  ROMARK      K

11.39.56.478        NETPUT  (031655)  HA =000315  TA =000374                               MSG NO. 000047
 ABT =01  ADR =0001  ABN =000077  ACT =04  STATUS = 00000000   TLC = 0020

     001 B4248504BB5CB6D 55022205011355345555   BREAK 1     4$  ; 6
     002 000000000000000 00000000000000000000               P

11.39.56.478        NETPUT  (031655)  HA =000315  TA =000374                               MSG NO. 000048
 ABT =02  ADR =0001  ABN =000078  ACT =04  STATUS = 00000000   TLC = 0020

     001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000               0

11.39.56.960        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000049
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001000000 40601000000100000000  FCACK

11.39.56.964        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000050
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001001340 40601000000100011500  FCACK            4

11.39.56.992        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000051
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001001380 40601000000100011600  FCACK            8

11.39.57.021        NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010       MSG NO. 000052
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0000
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 7 of 13)

```
11.39.57.027        NETPUT   (031655)   HA =000315   TA =000374                          MSG NO. 000053
 ABT =01  ADR =0001  ABN =000079  ACT =03  STATUS = 00001000   TLC = 0001

     001  01F000000000000  00370000000000000000    4

11.39.57.028        NETPUT   (031655)   HA =000315   TA =000374                          MSG NO. 000054
 ABT =02  ADR =0001  ABN =000080  ACT =04  STATUS = 00000000   TLC = 0020

     001  B49390554B50313  55111620252455201423    INPUT PLS  4  UKP1
     002  000000000000000  00000000000000000000               0

11.39.57.501        NETGETL  (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063    MSG NO. 000055
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001  8302000010013C0  40601000000100011700  FCACK            <

11.39.57.505        NETGETL  (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063    MSG NO. 000056
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001  830200001001400  40601000000100012000  FCACK            @

11.40.12.998        NETGETL  (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010    MSG NO. 000057
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000   TLC = 0005

     001  04504E04404304E  01050116010401030116  AEANADACAN  ENDCN

11.40.13.005        NETPUT   (031655)   HA =024544   TA =024545                          MSG NO. 000058
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002

     001  630600001000000  30603000000100000000  CONEND      C
     002  2411ADB6DB40000  11010655555555000000  IAF          A [M4

11.40.13.064        NETGETL  (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063    MSG NO. 000059
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001  634600001000000  30643000000100000000  CONENDN     CF

11.40.29.864        NETGETL  (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063    MSG NO. 000060
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0010

     001  630000001600200  30600000000130001000  CONREQ      C
     002  51C75F0ADB45018  24343537025555050030  T124B  E X    UP-4P
     003  0000000000006EA  00000000000000003352        0)        N
     004  0000000002DD40B  00000000000013352013      K2PK       -T
     005  xxxxxxx6DB40011  xxxxxxxxxx5555000021  xxxxx    Q  M B [@
     006  xxxxxxxE1880037  xxxxxxxxxxxxxx000067  xxxxxxx  &    16A  7
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 8 of 13)

```
     007 000FF8FFFFFFFFF 00007770777777777777    ;';;;;;;    X____
     008 FFF3400001FFFFF 77771500000007777777  ;;M   G;;;   _4
     009 000000000000F6F 00000000000000007557          .       V
     010 7C014034460D1C1 37000500150430150701  4 E MDXMGA  W@  D@QA

11.40.29.870       NETPUT  (031655)  HA =024544   TA =024545                  MSG NO. 000061
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 634000001000C1 30640000000100000301  CONREQN      C@

11.40.30.922       NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000062
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830700001000000 40603400000100000000  FCINIT

11.40.30.925       NETPUT  (031655)  HA =024544   TA =024545                  MSG NO. 000063
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 834700001000000 40643400000100000000  FCINITN      G

11.40.30.925       NETPUT  (031655)  HA =000315   TA =000374                  MSG NO. 000064
 ABT =01  ADR =0001  ABN =000081  ACT =04  STATUS = 00000000   TLC = 0020

     001 71235676D58549E 34221526355526052236  1RMV2 VER3  Q#VVU I
     002 000000000000000 00000000000000000000              @

11.40.30.925       NETPUT  (031655)  HA =000315   TA =000374                  MSG NO. 000065
 ABT =02  ADR =0001  ABN =000082  ACT =04  STATUS = 00000000   TLC = 0020

     001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
     002 000000000000000 00000000000000000000              0

11.40.31.468       NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000066
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001001440 40601000000100012100  FCACK        ^     D

11.40.31.473       NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000067
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001

     001 830200001001480 40601000000100012200  FCACK              H

11.41.39.064       NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010     MSG NO. 000068
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10000000   TLC = 0100
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 9 of 13)

```
11.41.39.077        NETGET  (031340)  ACN =0001  HA =000315  TA =000374  TLMAX =0063       MSG NO. 000069
 ABT =01  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000  TLC = 0100

     001 054068069073020 01240150015101630040  ATA/A(A% 5  THIS
     002 069073020061020 01510163004001410040  A(A% 5A6 5  IS A
     003 074065073074020 01640145016301640040  A"A+A%A" 5  TEST
     004 06F066020074068 01570146004001640150  A.A- 5A"A/  OF TH
     005 065020071075065 01450040016101650145  A+ 5A[A_A+  E QUE
     006 07506906E067020 01650151015601470040  A_A(A,A* 5  UING
     007 06306F064065020 01430157014401450040  A8A.A9A+ 5  CODE
     008 06606F07202006D 01460157016200400155  A-A.A] 5A   FOR M
     009 065073073061067 01450163016301410147  A+A%A%A6A*  ESSAG
     010 06507302006F066 01450163004001570146  A+A% 5A.A-  ES OF
     011 02006D06F072065 00400155015701620145   5A A.A]A+   MORE
     012 02007406806106E 00400164015001410156   5A"A/A6A,   THAN
     013 02006F06E065020 00400157015601450040   5A.A,A+ 5   ONE
     014 06E06507407706F 01560145016401670157  A,A+A"A&A.  NETWO
     015 07206B020064061 01620153004001440141  A]A$ 5A9A6  RK DA
     016 07406102006206C 01640141004001420154  A"A6 5A7A=  TA BL
     017 06F06306B03B020 01570143015300730040  A.A8A$ > 5  OCK;
     018 074068069073020 01640150015101630040  A"A/A(A% 5  THIS
     019 06906E070075074 01510156016001650164  A(A,A#A_A"  INPUT
     020 02007306806F075 00400163015001570165   5A%A/A.A_   SHOU

11.41.39.083        NETPUT  (031655)   HA =000315   TA =000374                                  MSG NO. 000070
 ABT =01  ADR =0001  ABN =000083  ACT =03  STATUS = 00001000  TLC = 0101

     001 054068069073020 01240150015101630040  ATA/A(A% 5  THIS
     002 069073020061020 01510163004001410040  A(A% 5A6 5  IS A
     003 074065073074020 01640145016301640040  A"A+A%A" 5  TEST
     004 06F066020074068 01570146004001640150  A.A- 5A"A/  OF TH
     005 065020071075065 01450040016101650145  A+ 5A[A_A+  E QUE
     006 07506906E067020 01650151015601470040  A_A(A,A* 5  UING
     007 06306F064065020 01430157014401450040  A8A.A9A+ 5  CODE
     008 06606F07202006D 01460157016200400155  A-A.A] 5A   FOR M
     009 065073073061067 01450163016301410147  A+A%A%A6A*  ESSAG
     010 06507302006F066 01450163004001570146  A+A% 5A.A-  ES OF
     011 02006D06F072065 00400155015701620145   5A A.A]A+   MORE
     012 02007406806106E 00400164015001410156   5A"A/A6A,   THAN
     013 02006F06E065020 00400157015601450040   5A.A,A+ 5   ONE
     014 06E06507407706F 01560145016401670157  A,A+A"A&A.  NETWO
     015 07206B020064061 01620153004001440141  A]A$ 5A9A6  RK DA
     016 07406102006206C 01640141004001420154  A"A6 5A7A=  TA BL
     017 06F06306B03B020 01570143015300730040  A.A8A$ > 5  OCK;
     018 074068069073020 01640150015101630040  A"A/A(A% 5  THIS
     019 06906E070075074 01510156016001650164  A(A,A#A_A"  INPUT
     020 02007306806F075 00400163015001570165   5A%A/A.A_   SHOU
     021 01F000000000000 00370000000000000000   4

11.41.42.759        NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063       MSG NO. 000071
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 10 of 13)

```
    001 830200001 0014C0 4060100000010001 2300  FCACK            L

11.41.42.791        NETGETL  (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010      MSG NO. 000072
 ABT =00  ADR =0001  ABN =000000  ACT =02  STATUS = 10010000  TLC = 0070

11.41.42.823        NETGET   (031340)  ACN =0001  HA =000315  TA =000374  TLMAX =0063      MSG NO. 000073
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00010000  TLC = 0070

    001 06C064020067065 01540144004001470145  A=A9 5A*A+  LD GE
    002 06E065072061074 01560145016201410164  A,A+A]A6A"  NERAT
    003 065020073065076 01450040016301450166  A+ 5A%A+A!  E SEV
    004 06507206106C020 01450162014101540040  A+A]A6A= 5  ERAL
    005 06206C06F06306B 01420154015701430153  A7A=A.A8A$  BLOCK
    006 07302006F066020 01630040015701460040  A% 5A.A- 5  S OF
    007 06906E070075074 01510156016001650164  A(A,A#A A"  INPUT
    008 02006106E064020 00400141015601440040   5A6A,A9 5   AND
    009 06F075074070075 01570165016401600165  A.A A"A#A   OUTPU
    010 07402006106E064 01640040014101560144  A" 5A6A,A9  T AND
    011 020062065020070 00400142014500400160   5A7A+ 5A#   BE P
    012 07206F070065072 01620157016001450162  A]A.A#A+A]  ROPER
    013 06C079020065063 01540171004001450143  A=A? 5A+A8  LY EC
    014 06806F06506402E 01500157014501440056  A/A.A+A9 ,  HOED.

11.41.42.843        NETPUT   (031655)   HA =000315   TA =000374                           MSG NO. 000074
 ABT =01  ADR =0001  ABN =000084  ACT =03  STATUS = 00001000  TLC = 0071

    001 06C064020067065 01540144004001470145  A=A9 5A*A+  LD GE
    002 06E065072061074 01560145016201410164  A,A+A]A6A"  NERAT
    003 065020073065076 01450040016301450166  A+ 5A%A+A!  E SEV
    004 06507206106C020 01450162014101540040  A+A]A6A= 5  ERAL
    005 06206C06F06306B 01420154015701430153  A7A=A.A8A$  BLOCK
    006 07302006F066020 01630040015701460040  A% 5A.A- 5  S OF
    007 06906E070075074 01510156016001650164  A(A,A#A A"  INPUT
    008 02006106E064020 00400141015601440040   5A6A,A9 5   AND
    009 06F075074070075 01570165016401600165  A.A A"A#A   OUTPU
    010 07402006106E064 01640040014101560144  A" 5A6A,A9  T AND
    011 020062065020070 00400142014500400160   5A7A+ 5A#   BE P
    012 07206F070065072 01620157016001450162  A]A.A#A+A]  ROPER
    013 06C079020065063 01540171004001450143  A=A? 5A+A8  LY EC
    014 06806F06506402E 01500157014501440056  A/A.A+A9 ,  HOED.
    015 01F000000000000 00370000000000000000   4

11.41.42.843        NETPUT   (031655)   HA =000315   TA =000374                           MSG NO. 000075
 ABT =02  ADR =0001  ABN =000085  ACT =04  STATUS = 00000000  TLC = 0020

    001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
    002 000000000000000 00000000000000000000              0

11.41.43.280        NETGETL  (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063      MSG NO. 000076
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 11 of 13)

```
ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 830200001001500 40601000000100012400  FCACK          P

11.41.43.284      NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000077
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 830200001001540 40601000000100012500  FCACK          T

11.42.12.987      NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010     MSG NO. 000078
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000010  TLC = 0037

    001 04E06F077020074 01160157016700400164  ANA.A& 5A"  NOW T
    002 06F020074065073 01570040016401450163  A. 5A"A+A%  O TES
    003 074020074068065 01640040016401500145  A" 5A"A/A+  T THE
    004 02006906E070075 00400151015601600165   5A(A,A#A    INPU
    005 07402006306106E 01640040014301410156  A" 5A8A6A,  T CAN
    006 06306506C06906E 01430145015401510156  A8A+A=A(A,  CELIN
    007 06702006306F064 01470040014301570144  A* 5A8A.A9  G COD
    008 065040000000000 01450100000000000000  A+A         E@

11.42.13.003      NETPUT  (031655)  HA =000315  TA =000374                             MSG NO. 000079
 ABT =02  ADR =0001  ABN =000086  ACT =04  STATUS = 00000000  TLC = 0020

    001 B49390554B50313 55111620252455201423   INPUT PLS  4  UKP1
    002 000000000000000 00000000000000000000              0

11.42.14.014      NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000080
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001

    001 830200001001580 40601000000100012600  FCACK          X

11.42.18.844      NETGETL (031354)  ALN =0001  HA =000315  TA =000374  TLMAX =0010     MSG NO. 000081
 ABT =02  ADR =0001  ABN =000000  ACT =03  STATUS = 00000000  TLC = 0006

    001 053048055054044 01230110012501240104  ASAHAUATAD  SHUTD
    002 04E000000000000 01160000000000000000  AN          N

11.42.18.860      NETPUT  (031655)  HA =024544  TA =024545                             MSG NO. 000082
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0002

    001 630600001000000 30603000000100000000  CONEND      C
    002 2411ADB6DB40000 11010655555555000000  IAF          A [M4

11.42.18.927      NETGETL (031354)  ALN =0000  HA =024544  TA =024545  TLMAX =0063     MSG NO. 000083
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 12 of 13)

```
                                   RMV2    LOG FILE OUTPUT                              83/08/05
                                   DATE RECORDED - 83/08/05                             PAGE 00013

    001 634600001000000 306430000001000000000  CONENDN     CF

11.42.26.021         NETOFF   (030077)       DATE =83/08/05                       MSG NO. 000084
```

Figure 7-4. Debug Log File Listing for Sample FORTRAN Program (Sheet 13 of 13)

```
NAM STATISTICS GATHERING STARTED
NETON   DATE  83/08/05.   TIME 11.38.26.

NAM STATISTICS GATHERING TERMINATED
NETOFF  DATE  83/08/05.   TIME 11.42.26.

CPU TIME USED:            0.244 SEC


NUMBER OF PROCEDURE CALLS
    NETGET                  2
    NETGETL                46
    NETPUT                 34
    NETWAIT                47

NUMBER OF WORKLIST TRANSFER ATTEMPTS
    SUCCESSFUL             64

NUMBER OF INPUT/OUTPUT BLOCKS TRANSFERRED
    INPUT   ABT=0           2
    INPUT   ABT=1           1
    INPUT   ABT=2           8
    INPUT   ABT=3          37
    OUTPUT  ABT=1          11
    OUTPUT  ABT=2          11
    OUTPUT  ABT=3          12

NUMBER OF ERRORS
```

Figure 7-5. Statistical File Listing for Sample FORTRAN Program

QUEUED TERMINAL RECORD MANAGER

The Queued Terminal Record Manager (QTRM) utility package allows an application program to use NAM to perform input and output to and from a device 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 section 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 device. The program then has a connection with a terminal or an application. When the term connection is used in this section, it refers to the general case and includes both device-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 supports.

The network information table has the format shown in figure 8-1. 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 only 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 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 device or application is assigned to connection number 3, QTRM writes all information concerning that device 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 device 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 device 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 device 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 device 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 device or application serviced cumulatively during a single program execution. The network reassigns a connection number when a device or application disconnects from the program, so that several devices 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 devices at the same time, it provides eight 10-word entries. During a single execution, six different devices might use each of those entries in succession, but each device 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.

	Word	59	53	47	35	29	17	11	5	0
net-info-table										
Global Entry for QTRM communication	1	application-name C(7)					char-set I(6)	num-conns I(12)		Read and write portion, occurs only once
	2	NAM-supervisor-word								
	3	reserved for CDC								
	4	reserved for CDC					sub-system code		Ato-A	
	5	max-trans-size I(12)		current-trans-size I(12)	sleep I(6)	connection-number I(12)	return-code I(6)	†	int-msg I(6)	
	6	next-application-name C(7)					xsleep I(18)			
	7	requested-application-name C(7)					destination-host C(3)			
	8	reserved for CDC								
	9	reserved for CDC								
	10	reserved for installation use								
Entry for connection 1	1	terminal-name-1/application-name-1 C(7)					tclass-1 I(6)	page-width-1 I(12)		Read-only portion, repeated once for each connection
	2	family-name-1 C(7)					dev-type I(6)	page-length-1 I(12)		
	3	user-name-1 C(7)					res	max-block-size-I(12)		
	4	abl-1 I(6)	current-abn-1 I(18)		acknowledged-abn-1 I(18)		state-1 I(6)	res	current-abl-I(6)	
	5	reserved for CDC								
	6	upline-abh-1								
	7	downline-abh-1								
	8	reserved for CDC								
	9	reserved for CDC								
	10	reserved for installation use								
		⋮								
Entry for connection n (n=num-conns)	1	terminal-name-n/application-name-n					tclass-n	page-width-n		
	2	family-name-n					dev-type I(6)	page-length-n		
	3	user-name-n					res	max-block-size-n		
	4	abl-n	current-abn-n		acknowledged-abn-n		state-n	res	current-abl-n	
	5	reserved for CDC								
	6	upline-abh-n								
	7	downline-abh-n								
	8	reserved for CDC								
	9	reserved for CDC								
	10	reserved for installation use								

†sec-return-code I(6)

Figure 8-1. Network Information Table Format (Sheet 1 of 10)

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 by other application programs or terminal users to access 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 normalized 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 normalized 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 devices 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 2.
num-conns	This field contains a 12-bit integer, 1 $\le$ num-conns $\le$ 4095, indicating how many connections the application program can simultaneously support. Connections are assigned numbers from 1 to num-conns; the value used for numconns should not be greater than the number of 10-word entries provided in the network information table. The network information table must be 10+(10 X 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 8-1. Network Information Table Format (Sheet 2 of 10)

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.)
sub return code	This 12-bit field contains the reason code returned in the CON/ACRQ/A supervisory message. The field has meaning only when the return code field has the value 13. The reason codes for the supervisory message are explained in section 3.
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 char-set = 1, one max-trans-size unit = 60 bits. If char-set = 2 or 10, one max-trans-size unit = 8 bits. If char-set = 3 or 11, one max-trans-size unit = 12 bits. If char-set = 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 char-set = 1, one current-trans-size unit = 60 bits. If char-set = 2 or 10, one current-trans-size unit = 8 bits. If char-set = 3 or 11, one current-trans-size unit = 12 bits. If char-set = 4, one current-trans-size unit = 6 bits.

Figure 8-1. Network Information Table Format (Sheet 3 of 10)

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 Display-Code Output Editing. Current-trans-size must be less than or equal to max-trans-size.

sleep

This 6-bit field contains a signed integer that tells QTRM what action to take after the application program issues a QTGET call. (See also the XSLEEP field.) This field can have the values:

-n Where $1 \leq n < 32$; if no data block or return-code field value other than 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 Interrogate XSLEEP to determine what action to take after QTGET is issued.

+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 after QTOPEN has been called. (QTOPEN sets the field to zero.)

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.

xsleep

This 18-bit field contains a signed integer that tells QTRM what action to take after the application program issues a QTGET call. (See also the SLEEP field.) This field can have the values:

-n Where $1 \leq n < 4096$; if no data block or return-code field value other than 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 field 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 $1 \leq n < 4096$; 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.

Figure 8-1. Network Information Table Format (Sheet 4 of 10)

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. 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 or xsleep field contained 0 or a positive value.

2 A new device 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 device, 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 device were discarded by the network.

4 Reserved for CDC use.

5 The current-abl value for the connection identified in the connection-number field has been exceeded. This return-code value only occurs after a QTPUT call is attempted when the current-abl value for the connection 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 connection to a nonzero value.

6 The connection between NAM and the device 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 device or application program was lost by the network.

A block to or from the device or application program was too long to deliver.

The terminal sent transparent data to the program.

The other application program terminated or ended the connection.

No additional communication is possible between the application program and that device 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.

Figure 8-1. Network Information Table Format (Sheet 5 of 10)

7	The user at the terminal identified in the connection-number field has entered a user-break-1 character or caused a user-break-1 condition. This value only occurs after a QTGET call. On return from the call, QTRM has reset the current-abl field for the affected device to the value in the device abl field; this change indicates that any blocks previously sent by the program but not yet delivered to the device 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 device to the value in the device abl field; this change indicates that any blocks previously sent by the program but not yet delivered to the device 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.
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 The reason codes for these supervisory messages are explained in section 3.
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 62	Reserved for CDC use.
63	An internal or uncoded error. If this happens, it means something severe has taken place in QTRM. You should close your files, abort your program, and do a dump.

Figure 8-1. Network Information Table Format (Sheet 6 of 10)

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. If the supervisory message is a CON/ACRQ/A, this field contains the value of subfield rc2 from the supervisory message.
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 only has significance to the destination application. 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.
next-application-name	This 42-bit character data field contains the network application program name identifying the program to which a device should be switched during processing of a QTENDT call. This field can contain the following: 0 — The network software uses prompting dialog or automatic login information to determine the next application program the device communicates with, or disconnects the device from the host if that is an appropriate action. NVF command — The Network Validation Facility reinitiates the login sequence for the device or causes terminal disconnection from the host. valid program name — The device 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 device to be disconnected from the host. HELLO / LOGIN — Reinitiate login for the device; 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. For an application-to-application connection, this field must contain a 0. 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 under Terminating Connections in section 3. This field is not used with QTENDT calls for application-to-application connections.

Figure 8-1. Network Information Table Format (Sheet 7 of 10)

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. This is the first identifier for the connection. This identifier can be one to seven letters or digits long and is left-justified with blank fill within this field; the first character must be a letter. For intra-host connections, this field contains the name of the application program with which your program needs to establish a connection. For inter-host connections, the name you use must match the value of the NAME1 parameter in the NDL OUTCALL statement used by your program.
destination-host	This 18-bit character data field contains the second identifier for a connection your program initiates with a QTLINK call. If the connection is between two hosts, this identifier must be one to three letters or digits, left-justified with blank fill within the field; the first character must be a letter. If the connection is within a host, this identifier can be a binary 0. By convention, any nonzero name is the name of the destination host in which the other application program runs. The name you use must match the value of the NAME2 parameter in the NDL OUTCALL statement used by your program.
terminal-name-i/ application-name-i	This 42-bit character data field contains the display code characters of the name used to identify the device 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 device. For an application-to-application connection, this field contains blanks.
tclass-i	This 6-bit field contains the integer terminal class associated by the network with the device 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 device on connection i. This field is not used for application-to-application connections.
page-width-i	This 12-bit field contains the integer page width value associated by the network with the device 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 device on connection i. This field is not used for 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 device connection i. The family name is one to seven letters or digits long and is left-justified with blank fill within this field. This field is not used for application-to-application connections.
dev-type-i	This 6-bit field contains an integer value to identify the type of connection for connection i. The integer used in this field is one of those described for the dt field of the connection-request supervisory messages presented in section 3. Typical values are: 0 — This connection is a device-to-application connection for a console. 5 — This connection is an application-to-application connection within the same host. 6 — This connection is an application-to-application connection between hosts. 12 thru 15 — This connection is a device-to-application connection for a device with a site-defined device type.

Figure 8-1. Network Information Table Format (Sheet 8 of 10)

page-length-i
: This 12-bit field contains the integer page length value associated by the network with the device 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 device on connection i.

  This field is not used for 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 device connection i. The user name is one to seven letters, digits, or asterisks long and is left-justified with blank fill within the field.

  This field is not used for application-to-application connections.

res
: Reserved by CDC.

max-block-size-i
: This 12-bit field contains the integer downline block size in character units for the device on connection i. This block size is based on the network configuration file information for the device or the local configuration file information for an application-to-application connection. The block size is a suggested value for adjusting the current-trans-size field based on efficiency considerations for the site.

abl-i
: This 6-bit integer field contains the number of 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 device-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.

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 device 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 connection).

Figure 8-1. Network Information Table Format (Sheet 9 of 10)

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 8-1. Network Information Table Format (Sheet 10 of 10)

In figure 8-1, 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.

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 8-1 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 8-2.

```
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 8-1. The fields in the network information table must be initialized before the call to QTOPEN is issued.

Figure 8-2. 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-CONNS 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. RMV2 supports five devices simultaneously on connections numbered 1 through 5, uses 6-bit display code for all input and output transmissions, and cannot process transparent mode 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 device or application requests access to it. As soon as a device 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 device 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 8-3.

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 device 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 normalized mode data in the text area used by any NETPUT call to AIP. These constraints are described in section 2.

Figure 8-3. 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 device in terminal class 4 correctly when output queuing is performed. Devices in that class, such as the 2741, have lockable keyboards. When output begins, the network software locks the device 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 device is permitted until input is completed. If a message comprising nine blocks is being sent to the device, 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 device-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 device-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 current-trans-size field 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 device-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 connection, 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 device 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

25 The application program has received the reset response from the network for the connection identified in the connection-number field. This value occurs only after a QTGET call and only for a connection that was in a break condition. The break condition exists only after the application program has called QTSUP to send a break condition to the other end of the connection. The application program can now call QTSUP to send another break condition for this connection.

26 The application program has received a priority data message (INTR/USR/R supervisory message with reason code other than user break 1 or 2) on the connection identified in the connection-number field. This value occurs only after a QTGET call and only if the application program called QTCMD to request notification of user interrupts. The network does not require any action or response from the application program. QTRM sends the user acknowledgment (INTR/RSP/N supervisory message) for the interrupt connection. The sub-return-code field contains the reason code for the interrupt condition.

27 The application program has received the user interrupt response (INTR/RSP/R supervisory message) on the connection identified in the connection-number field. This value occurs only after a QTGET call and only if the application program called QTSUP to send a user interrupt to the other end of the connection. The network does not require any action or response from the application program. The application program can now call QTSUP to send another user interrupt for this connection.

28 The application program has received the user break marker (BI/MARK/R supervisory message) on the connection identified in the connection-number field. This value occurs only after a QTGET call and only if the application program previously called QTCMD to indicate that it wanted to be informed about the user break marker. The connection involved is always a device connection in a user break condition (return code 7 or 8 response for this connection in a previous QTGET call). The application program must call QTTIP to send a RO/MARK/R synchronous supervisory message before any more downline data will be delivered to the device. After receiving this return code, the application program assumes that any new data it receives on the connnection was entered after the user break.

29 The application program has received a synchronous supervisory message other than BI/MARK/R (user break mark) on the connection identified in the connection-number field. This value occurs only after a QTGET call. The connection involved is always a device connection. The synchronous supervisory message is in the application program text input area identified for the QTGET call; the message originates from either CCP or CDCNET. The network does not require any action or response from the application program.

30 Reserved for CDC use.

31 The host operator has assigned the NAM K-display to this application program. This value occurs only after a QTGET call when the application program has previously called QTCMD to inform QTRM that it supports the NAM K-display. If the text input area is at least one word long, QTRM stores the length of the left and right screens in the first word of the text input area. This first word is actually the first word of the HOP/START/R supervisory message. The format of this message is described in section 3. If the text input area is not large enough, the information on the size of the left and right screens is lost.

Upon receipt of this return code, the application program should generate a banner or other display data and call QTSUP to send the display data to NAM.

Figure 8-1. Network Information Table Format (Sheet 12 of 24)

32	The host operator has entered a K-display typein. This value occurs only after a QTGET call when the application program has been assigned the NAM K-display (return code 31 was received after a QTGET call). QTRM writes the operator typein to the application program text input area identified for that QTGET call. If the text input area is not large enought to contain the operator typein, QTRM truncates the typein and the last part of the typein is lost. The current-trans-size field contains the number of characters (in display code only) of the operator typein written to the text input area. Until the application program sends display data back to NAM with the input-allowed flag set (QTSUP call with parm-flag1 set to 1), the host operator cannot enter any more NAM K-display typeins for this application program.
33	The host operator has entered the break character. This value occurs only after a QTGET call and the application program has been assigned the NAM K-display (return code 31 was received after a QTGET call). The operator is probably no longer interested in data from this application program and wants to enter another typein. The application program should call QTSUP to send a HOP/DIS/R supervisory message to NAM with the input-allowed flag set (parm-flag1 set to 1 for the QTSUP call). No K-display data is necessary for this call (current-trans-size field is set to zero).
34	The host operator has entered a page character. This value occurs only after a QTGET call when the application program has been assigned the NAM K-display (return code 31 was received after a QTGET call). Because the host console can display only a finite number of lines of data, NAM scrolls the oldest data off the top of the screen. The newest data always appears at the bottom of the screen. If the application program sends more than one screen of display data, only the last part of the data remains on the screen. This may or may not be desired by the operator. The page character allows the operator to inform the application program whether it should send all the display data at once or only one screen at a time. There are four page characters: the plus (+) character, the minus (-) character, the left parenthesis (() character, and the right parenthesis ()) character. The plus and minus characters control paging of the left screen. The left and right parenthesis characters control paging of the right screen. If the page character is the plus character, the operator is requesting the application program to page the left screen display data one screen at a time. If the page character is the minus character, the operator is requesting the application program not to page the left screen display data. If the page character is the left parenthesis character, the operator is requesting the next page of the right screen display data. If the page character is the right parenthesis character, the operator is requesting the previous page of the right screen display data. For the left screen, the convention is that all application programs should assume that paging is off when the operator assigns the K-display to them. The application program should never send more than one screen full of display data to the left screen at one time. If the application program has more than one screen to send, it must wait for another plus page character before sending the next screen (after a QTGET call, the application program must receive another return code 34 with the page character set to the plus character). This interaction between the operator and the application program can be repeated until the application program has no more data to display or the operator enters another command.

Figure 8-1. Network Information Table Format (Sheet 13 of 24)

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 device.

Check the connection entry in the network information table to determine whether 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 connection 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 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 connection 4, so that no confusion exists if another device or application program is assigned to the same number.

ENDING COMMUNICATION WITH THE NETWORK (QTCLOSE)

The application program can end communication with all connected devices or applications and with the network software simultaneously by calling QTCLOSE. This call has the format shown in figure 8-7.

```
ENTER FORTRAN-X QTCLOSE
```

Figure 8-7. 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 (devices receive 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 devices and applications still connected to the program

Determine that all transmitted blocks have been delivered to the connection

Issue QTENDT calls for all remaining device 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 section. Examples of QTCLOSE calls appear in several contexts within the program at the end of this section.

OUTPUT FORMATTING AND EDITING

Output transmitted through QTRM to a device 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 device (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 device through QTRM begins with a format effector character. The format effector character controls placement of the line on the device 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.

DISPLAY-CODE OUTPUT EDITING

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 connections 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 8-8. 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 connection entry portion of the table. The following paragraphs explain the logic used to design the algorithm.

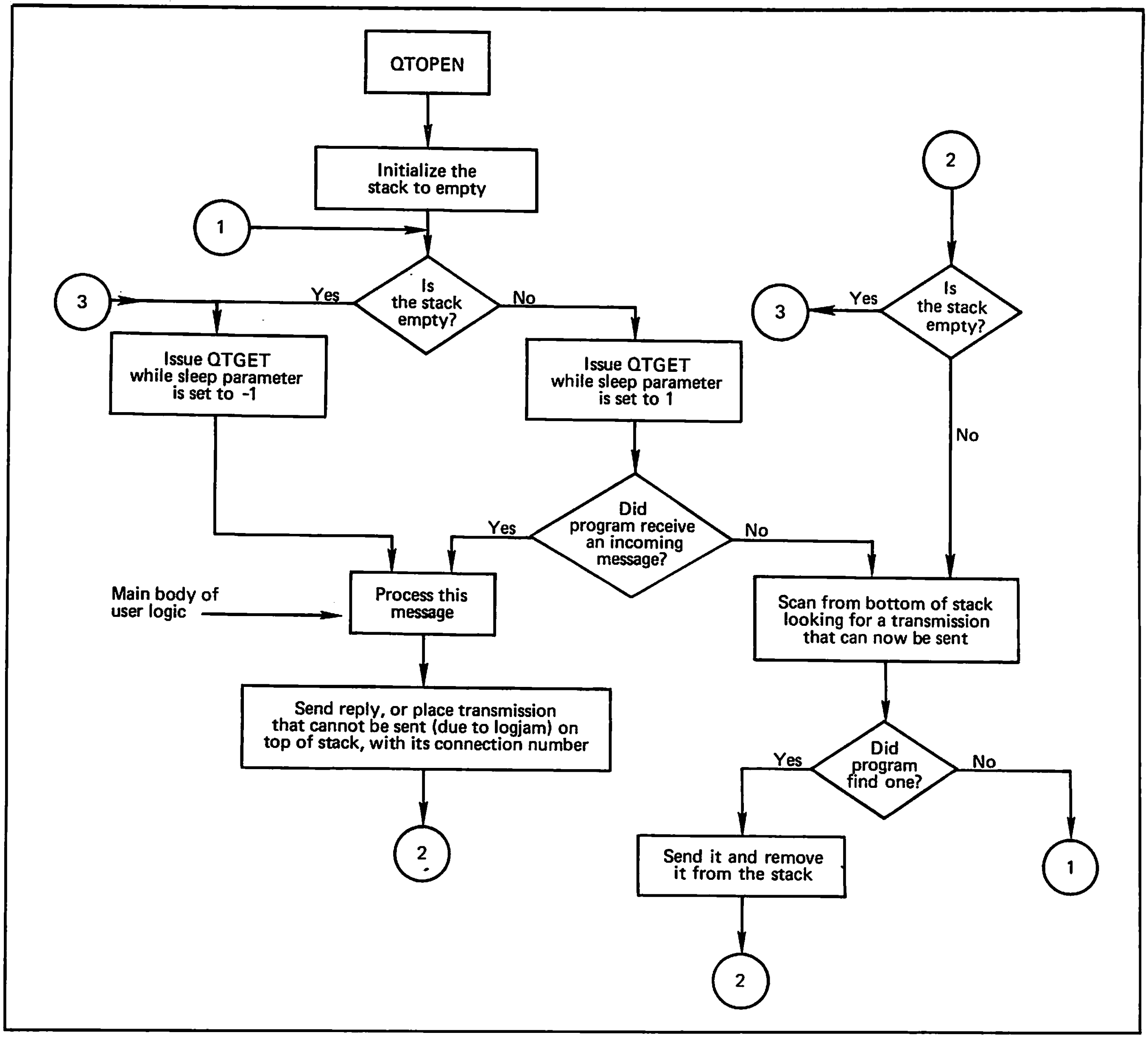


Figure 8-8. Algorithm for Output Buffering Using QTRM

When an application program services only one connection, the network can be made to cope with situations where the program produces output faster than a device 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 device catches up with the program.

You cannot use the sleep parameter as a solution when the application program services more than one connection because the program is always rolled back in when input is available from any connection. Thus, input from device B brings the program back into central memory even though the output backlog for device A has not disappeared. A program servicing several connections always requires an output queuing algorithm that applies to each, when each connection potentially can be sent more than one block in a single message.

Programs can also be coded for the opposite (typeahead) 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. Typeahead 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 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 connection 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 connection by checking the current-abl field of the connection entry in the network information table before each QTPUT call involving that connection. 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 connection. 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 device 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 connection in QTPUT calls involving one or more lines in blocks up to the max-block-size value for the connection 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 connection. This approach requires a small input queue for each connection, 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 8-9 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 section 7. Both programs return to the terminal user each block entered from the device. Both programs queue output blocks and permit a prompting message to be output after each returned message. 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 588   SOURCE LISTING OF ECHO-RM   AOPT= 66/CDC/CDCS2   83/06/16. 12.21.30.   PAGE  1

               IDENTIFICATION DIVISION.
               PROGRAM-ID. ECHO-RMV2.
               ENVIRONMENT DIVISION.
               CONFIGURATION SECTION.
               INPUT-OUTPUT SECTION.
               FILE-CONTROL.
               DATA DIVISION.
               FILE SECTION.
               WORKING-STORAGE SECTION.
               01 NETWORK-INFORMATION-TABLE.
                   02 GLOBAL-PORTION.
                     03 APPLICATION-NAME PIC X(7).
                     03 CHARACTER-SET PIC 9 COMP-4.
                     03 NUMBER-CONNECTIONS PIC 999 COMP-4.
                     03 NAM-SUPERVISOR-WORD PIC X(10).
                     03 FILLER PIC X(19).
                    03 APPLICATION-TO-APPLICATION PIC 9 COMP-4.
              *
              *THE PICTURE SIZE USED FOR COMPUTATIONAL ITEMS SUCH AS
              *MAX-TRANS-SIZE AND SLEEP IS CHOSEN TO PERMIT STORAGE OF
              *THE LARGEST POSSIBLE FIELD VALUE WITHOUT TRUNCATION OF
              *THE VALUE DIGITS.
              *
                     03 MAX-TRANS-SIZE PIC 999 COMP-4.
                     03 MESSAGE-LENGTH PIC 999 COMP-4.
                     03 SLEEP PIC S9 COMP-4.
                     03 CONNECTION-NUMBER PIC 999 COMP-4.
                     03 RETURN-CODE PIC 9 COMP-4.
                     03 SECONDARY-RETURN-CODE PIC 9 COMP-4.
                     03 INTERMEDIATE-MESSAGE PIC 9 COMP-4.
                     03 NEXT-APPLICATION-NAME PIC X(7).
                     03 REQUESTED-APPLICATION-NAME PIC X(7).
                     03 DESTINATION-HOST PIC X(3).
                     03 FILLER PIC X(33).
                   02 TERMINAL-ENTRY OCCURS 5 TIMES.
                     03 TERMINAL-NAME PIC X(7).
                     03 TERMINAL-CLASS PIC 9 COMP-4.
                     03 PAGE-WIDTH PIC 999 COMP-4.
                     03 FAMILY-NAME PIC X(7).
                     03 DEVICE-TYPE PIC X.
                     03 PAGE-LENGTH PIC 999 COMP-4.
                     03 USER-NAME PIC X(7).
                     03 FILLER PIC X.
                     03 MAXIMUM-BLOCK-SIZE PIC 999 COMP-4.
                     03 ABL PIC 9 COMP-4.
                     03 CURRENT-ABN PIC 9(4) COMP-4.
                     03 ACKNOWLEDGED-ABN PIC 9(4) COMP-4.
                     03 STATE PIC 9 COMP-4.
                     03 FILLER PIC X.
                     03 CURRENT-ABL PIC 9 COMP-4.
                     03 FILLER PIC X(10).
                     03 UPLINE-ABH PIC X(10).
                     03 DOWNLINE-ABH PIC X(10).
                     03 FILLER PIC X(30).
0         COLUMN    1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 1 of 7)

```
CDC COBOL 5.3 - LEVEL 588   SOURCE LISTING OF ECHO-RM   AOPT= 66/CDC/CDCS2   83/06/16. 12.21.30.   PAGE  2

           01 INCOMING.
               02 COMMAND PIC X(20).
               02 REST-OF-DATA PIC X(80).
           01 OUTGOING.
               02 PRINT-CONTROL PIC X.
               02 OUT-MESSAGE PIC X(140).
           01 FOUND-FLAG PIC 9.
           01 QUEUE-SIZE PIC 99.
           01 HOLDING-QUEUE.
          *
          *THIS IS A PUSHDOWN QUEUE USED FOR  STORAGE OF THOSE
          *OUTPUT BLOCKS THE PROGRAM IS TEMPORARILY PREVENTED FROM SENDING
          *TO THE TERMINAL BECAUSE OF BLOCK LIMIT OR OTHER EVENTS IN THE
          *NETWORK.
          *
               02 QUEUE-ENTRY OCCURS 1 TO 60 TIMES DEPENDING ON QUEUE-SIZE
                  INDEXED BY INX-1 INX-2.
                   03 S-CONNECTION-NUMBER PIC 999 COMP-4.
                   03 S-MESSAGE PIC X(140).
                   03 S-INTERMEDIATE-MESSAGE PIC 9 COMP-4.



           PROCEDURE DIVISION.


           INITIALIZATION.
          *
          *HERE, THE NETWORK INFORMATION TABLE IS PRESET.
          *
               MOVE "RMV2" TO APPLICATION-NAME.
               MOVE 4 TO CHARACTER-SET.
               MOVE 120 TO MAX-TRANS-SIZE.
          *
          *THE FORMAT EFFECTOR CHARACTER "." CAUSES THE CURSOR TO
          *RETURN TO THE LEFT EDGE OF THE SCREEN OR PAGE
          *FOLLOWING THE CONTENTS OF OUT-MESSAGE. THIS ACTION
          *LEAVES THE CURSOR POSITIONED SO THAT THE USER CAN ENTER
          *A LINE EQUAL TO THE FULL PAGE WIDTH OF THE TERMINAL.
          *

               MOVE "." TO PRINT-CONTROL.
               MOVE SPACES TO OUT-MESSAGE.
               MOVE SPACES TO INCOMING.
               MOVE 5 TO NUMBER-CONNECTIONS.
               MOVE -1 TO SLEEP.
               MOVE 1 TO INTERMEDIATE-MESSAGE.
               MOVE 0 TO QUEUE-SIZE.
               MOVE 0 TO APPLICATION-TO-APPLICATION.
               MOVE 0 TO FOUND-FLAG.
               ENTER FORTRAN-X QTOPEN USING NETWORK-INFORMATION-TABLE.

          *
          *ALL TERMINALS WILL BE SWITCHED AUTOMATICALLY TO IAF

      COLUMN    1         2         3         4         5         6         7         8
      12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 2 of 7)

```
CDC COBOL 5.3 - LEVEL 588   SOURCE LISTING OF ECHO-RM   AOPT= 66/CDC/CDCS2   83/06/16. 12.21.30.   PAGE  3

      *WHEN THEY ARE DISCONNECTED FROM THIS PROGRAM.
      *
           MOVE "IAF" TO NEXT-APPLICATION-NAME.

       MAIN-LOOP.
           PERFORM RECEIVER THRU RECEIVE-EXIT.

           IF STATE (CONNECTION-NUMBER) = 1
              GO TO MAIN-LOOP.
           IF RETURN-CODE = 2
              MOVE 0 TO INTERMEDIATE-MESSAGE
              PERFORM DISPLAY-BANNER THRU BANNER-EXIT
              GO TO MAIN-LOOP.
           IF RETURN-CODE = 4
              PERFORM PUSH-DOWN-QUEUE
              GO TO MAIN-LOOP.
           IF RETURN-CODE = 6
              PERFORM CONNECTION-BROKEN-ROUTINE THRU CB-EXIT
              GO TO MAIN-LOOP.
           IF RETURN-CODE = 7 OR = 8
              PERFORM FLUSH-QUEUE
              MOVE 0 TO INTERMEDIATE-MESSAGE
              MOVE "." TO PRINT-CONTROL
              MOVE "NO ACTION TAKEN. NEXT ENTRY?" TO OUT-MESSAGE
              PERFORM SENDER THRU SEND-EXIT
              GO TO MAIN-LOOP.
           IF RETURN-CODE = 9
              GO TO WRAP-UP.
      *
      *TO SIMPLIFY THE PROGRAM, ONLY EXPECTED CONDITIONS ARE PROCESSED
      *BY THE PRECEDING CODE. ALL OTHER CONDITIONS CAUSE THE PROGRAM
      *TO PLACE A DIAGNOSTIC MESSAGE IN THE FILE CALLED OUTPUT (WITH
      *THE DISPLAY STATEMENT) AND SHUT DOWN. NO DIAGNOSTIC APPEARS AT
      *THE TERMINAL.
      *
           IF RETURN-CODE NOT = 0
              DISPLAY "PROGRAM BUG OR OTHER ERROR" RETURN-CODE " "
              SECONDARY-RETURN-CODE STOP RUN.

              MOVE "." TO PRINT-CONTROL.

      *
      *IF A TERMINAL USER ENTERS THE WORD END, THE USER IS
      *DISCONNECTED BUT THE PROGRAM CONTINUES TO SERVICE OTHER
      *TERMINALS.
      *
           IF COMMAND = "END"
              PERFORM END-CONNECTION THRU EC-EXIT
              GO TO MAIN-LOOP.
      *
      *IF A TERMINAL USER ENTERS THE WORD SHUTDOWN, THE USER IS
      *DISCONNECTED AND THE PROGRAM SHUTS DOWN.
      *
           IF COMMAND = "SHUTDOWN"

COLUMN    1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 3 of 7)

```
             MOVE 0 TO INTERMEDIATE-MESSAGE
             MOVE "." TO PRINT-CONTROL
             MOVE "BYE FOREVER!" TO OUT-MESSAGE
             PERFORM SENDER THRU SEND-EXIT

             GO TO WRAP-UP.

      *
      *THE FOLLOWING CODE BEGINS THE INPUT-ECHOING PORTION
      *OF THIS PROGRAM.
      *
           MOVE INCOMING TO OUT-MESSAGE
           MOVE 1 TO INTERMEDIATE-MESSAGE
           MOVE "." TO PRINT-CONTROL
           PERFORM SENDER THRU SEND-EXIT
      *
      *SEND PROMPT FOR NEXT LINE, WHICH ALSO ENDS PRESENT OUTPUT
      *MESSAGE TO THIS TERMINAL.
      *
           MOVE 0 TO INTERMEDIATE-MESSAGE
           MOVE "." TO PRINT-CONTROL
           MOVE "NEXT ENTRY?" TO OUT-MESSAGE
           PERFORM SENDER THRU SEND-EXIT
           GO TO MAIN-LOOP.
      *
      *THIS ENDS THE MAIN PROGRAM PORTION OF ECHO-RMV2. THE FOLLOWING
      *PARAGRAPHS COMPRISE THE SUBROUTINES USED BY THE MAIN PROGRAM.
      *

       RECEIVER.
           IF QUEUE-SIZE = 0
              MOVE -1 TO SLEEP
      *
      *THE NEXT LINE PREVENTS LEFTOVER CHARACTERS FROM THE END OF THE
      *LAST INPUT LINE FROM BEING INCLUDED IN THE TRANSFER OF THE
      *CURRENT (AND PRESUMABLY SHORTER) LINE.
      *
              MOVE SPACES TO INCOMING
              ENTER FORTRAN-X QTGET USING INCOMING
              GO TO RECEIVE-EXIT.
           MOVE 1 TO SLEEP
           MOVE SPACES TO INCOMING
           ENTER FORTRAN-X QTGET USING INCOMING.
           IF RETURN-CODE NOT = 1
              GO TO RECEIVE-EXIT
              ELSE NEXT SENTENCE.
       QUEUE-OUTPUT-CODE.
           MOVE 0 TO FOUND-FLAG.
           PERFORM QUEUE-SCAN VARYING INX-1 FROM 1 BY 1
              UNTIL FOUND-FLAG = 1 OR INX-1 EXCEEDS QUEUE-SIZE.
           IF FOUND-FLAG = 0
              GO TO RECEIVER
              ELSE NEXT SENTENCE.
           SET INX-1 DOWN BY 1.

COLUMN    1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 4 of 7)

```
CDC COBOL 5.3 - LEVEL 588    SOURCE LISTING OF ECHO-RM    AOPT= 66/CDC/CDCS2    83/06/16. 12.21.30.    PAGE  5

      *
      *THE REMAINING CODE ATTEMPTS TO REMOVE ALL
      *QUEUED OUTPUT FROM THE OUTPUT QUEUE, ONE ENTRY AT A
      *TIME, REGARDLESS OF CONNECTION NUMBER. EACH SEND
      *OPERATION IS FOLLOWED BY A RETURN TO THE POINT IN
      *THE PROGRAM WHERE STATUS UPDATES ARE OBTAINED.
      *
           MOVE S-INTERMEDIATE-MESSAGE (INX-1) TO INTERMEDIATE-MESSAGE.
           MOVE S-CONNECTION-NUMBER (INX-1) TO CONNECTION-NUMBER.
           IF STATE (CONNECTION-NUMBER) = 3 GO TO RECEIVE-EXIT.
           MOVE "." TO PRINT-CONTROL.
           MOVE S-MESSAGE (INX-1) TO OUT-MESSAGE.
           PERFORM QUEUE-COMPRESSION VARYING INX-2 FROM INX-1 BY 1
              UNTIL INX-2 = QUEUE-SIZE.
           SUBTRACT 1 FROM QUEUE-SIZE.
           PERFORM SENDER THRU SEND-EXIT.
           IF QUEUE-SIZE = 0
              GO TO RECEIVER
              ELSE GO TO QUEUE-OUTPUT-CODE.
       RECEIVE-EXIT.
           EXIT.


       QUEUE-SCAN.
           MOVE S-CONNECTION-NUMBER (INX-1) TO CONNECTION-NUMBER.
           IF CURRENT-ABL (CONNECTION-NUMBER) EXCEEDS 0
              MOVE 1 TO FOUND-FLAG.

       QUEUE-COMPRESSION.
           MOVE QUEUE-ENTRY (INX-2 + 1) TO QUEUE-ENTRY (INX-2).

       FLUSH-QUEUE.
           SET INX-1 INX-2 TO 1.
           PERFORM FLUSH-LOOP UNTIL INX-2 EXCEEDS QUEUE-SIZE.
           SET INX-1 DOWN BY 1.
           SET QUEUE-SIZE TO INX-1.

       FLUSH-LOOP.
           IF S-CONNECTION-NUMBER (INX-1) = CONNECTION-NUMBER
              SET INX-2 UP BY 1
           ELSE
              PERFORM CONDITIONAL-QUEUE-MOVE
              SET INX-1 INX-2 UP BY 1.
       CONDITIONAL-QUEUE-MOVE.
           IF INX-1 NOT = INX-2
              MOVE QUEUE-ENTRY (INX-2) TO QUEUE-ENTRY (INX-1).

       SENDER.
           IF CURRENT-ABL (CONNECTION-NUMBER) = 0
              PERFORM PUSH-DOWN-QUEUE GO TO SEND-EXIT.

      *
      *THE PROGRAM HAS QTRM SCAN BACKWARDS THROUGH THE MESSAGE
      *AREA AND TRUNCATE THE MESSAGE AUTOMATICALLY. THIS PROCEDURE

COLUMN   1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 5 of 7)

```
           *IS COMPARABLE TO THE ONE USED BY CYBER RECORD MANAGER FOR
           *Z-TYPE RECORDS.
           *
                MOVE 0 TO MESSAGE-LENGTH.
                ENTER FORTRAN-X QTPUT USING OUTGOING.
           *
           *IF NAM HAS STOPPED OUTPUT ON THE CONNECTION TEMPORARILY, OR IF
           *THE BLOCK LIMIT HAS BEEN EXCEEDED (AN EVENT THAT SHOULD NOT
           *HAPPEN) FOR THE CONNECTION, THE MESSAGE IS RETURNED TO THE
           *QUEUE FOR A LATER TRY.
           *
                IF RETURN-CODE  = 5 PERFORM PUSH-DOWN-QUEUE.
            SEND-EXIT.
                EXIT.


            PUSH-DOWN-QUEUE.
                ADD 1 TO QUEUE-SIZE.
                IF QUEUE-SIZE EXCEEDS 60 DISPLAY "QUEUE OVERFLOW ABORT"
                   PERFORM DUMPER VARYING INX-1 FROM 1 BY 1
                   UNTIL INX-1 EXCEEDS 60
                   STOP RUN.

                MOVE INTERMEDIATE-MESSAGE TO S-INTERMEDIATE-MESSAGE
                   (QUEUE-SIZE).
                MOVE CONNECTION-NUMBER TO S-CONNECTION-NUMBER (QUEUE-SIZE).
                MOVE OUT-MESSAGE TO S-MESSAGE (QUEUE-SIZE).

           *
           *THE FOLLOWING PROMPT IS  MANDATORY, BECAUSE QTRM DOES NOT
           *AUTOMATICALLY ISSUE A PROMPT TO A NEW
           *CONNECTION TO INITIALIZE THAT CONNECTION. THE FOLLOWING
           *PROMPT IS SENT BECAUSE GOOD PROGRAMMING PRACTICE
           *REQUIRES A NETWORK APPLICATION PROGRAM TO IDENTIFY ITSELF
           *TO A TERMINAL USER.
           *
            DISPLAY-BANNER.
                MOVE "." TO PRINT-CONTROL.
                MOVE "THIS IS RMV2 USING QTRM. ENTER SOMETHING." TO
                   OUT-MESSAGE.
                PERFORM SENDER THRU SEND-EXIT.
            BANNER-EXIT.
                EXIT.

           *
           *NO CALL TO QTENDT IS NECESSARY DURING THIS PROCESSING BRANCH,
           *BECAUSE QTRM AUTOMATICALLY CLEANS UP THE CONNECTION WHEN IT
           *RETURNS THE CONNECTION-BROKEN STATUS.
           *
            CONNECTION-BROKEN-ROUTINE.
                DISPLAY "CONNECTION BROKEN - TERMINAL USER HUNG UP "
                   CONNECTION-NUMBER
                DISPLAY " FAMILY " FAMILY-NAME (CONNECTION-NUMBER)

COLUMN   1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 6 of 7)

```
CDC COBOL 5.3 - LEVEL 588   SOURCE LISTING OF ECHO-RM   AOPT= 66/CDC/CDCS2   83/06/16. 12.21.30.   PAGE  7

                   DISPLAY " USER " USER-NAME (CONNECTION-NUMBER).
               CB-EXIT.
                   EXIT.


              *
              *THE WAIT-FOR-QUIET CALLS PROVIDE A DELAY LOOP FOR THE
              *NETWORK TO CLEAN UP ALL OUTSTANDING SUPERVISORY MESSAGE
              *TRAFFIC RELATED TO THE SHUTDOWN. WITHOUT THIS LOOP,
              *SOME TERMINAL CONNECTIONS WOULD RECEIVE AN
              *"APPLICATION FAILED" MESSAGE.
              *
               WRAP-UP.
                   PERFORM GRACEFUL-DISCONNECTS THRU GD-EXIT VARYING
                      CONNECTION-NUMBER
                      FROM 1 BY 1 UNTIL CONNECTION-NUMBER = 6.
                   ENTER FORTRAN-X QTCLOSE.
                   STOP RUN.

               GRACEFUL-DISCONNECTS.
                   IF STATE (CONNECTION-NUMBER) = 2 PERFORM FLUSH-QUEUE
                      MOVE 0 TO INTERMEDIATE-MESSAGE
                      MOVE "." TO PRINT-CONTROL
                      MOVE "SHUTDOWN COMING" TO OUT-MESSAGE
                      PERFORM SENDER THRU SEND-EXIT
                      ENTER FORTRAN-X QTENDT.
               GD-EXIT.
                   EXIT.

               END-CONNECTION.
                   PERFORM FLUSH-QUEUE
                   MOVE 0 TO INTERMEDIATE-MESSAGE
                   MOVE "." TO PRINT-CONTROL
                   MOVE "GOODBYE FOR NOW.." TO OUT-MESSAGE.
                   PERFORM SENDER THRU SEND-EXIT.
                   ENTER FORTRAN-X QTENDT.
               EC-EXIT.
                   EXIT.


               DUMPER.

                   DISPLAY S-CONNECTION-NUMBER (INX-1)
                      S-MESSAGE (INX-1).

        COLUMN    1         2         3         4         5         6         7         8
        12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 8-9. Sample Program ECHO-RMV2 Source Listing (Sheet 7 of 7)

Figure 8-10 shows the commands used to execute ECHO-RMV2. ECHO-RMV2 exists as a direct access source code file named RMV2.

Figure 8-11 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 section 7 for program RMV3 because both programs use essentially the same AIP routines for the same functions and support the same kind of dialog. Figure 8-12 contains a statistics file listing for ECHO-RMV2.

Figure 8-13 is a console printer listing for two sequential connections using ECHO-RMV2 during a single execution of that program. The listing includes program-generated messages and a console input message that is echoed back.

```
ATTACH,RMV2.
COBOL5,I=RMV2.
LDSET(LIB=NETIOD)
LGO.
REWIND,ZZZZZSN.
COPY,ZZZZZSN.
DLFP(I=0)
COPY,INPUT,QTRMEXP.
REWIND,QTRMEXP.
SAVE,QTRMEXP.
```

Figure 8-10. ECHO-RMV2 Job Commands

```
                              RMV2    LOG FILE OUTPUT                                       83/06/16
                              DATE RECORDED - 83/06/16                                      PAGE 00001

12.21.41.000        NETON   (004750)     ANAME = RMV2          DATE = 83/06/16              MSG NO. 000001
 NSUP ADDR = 001507  MINACN =00001  MAXACN =00005


12.21.41.039        NETPUT  (006634)   HA =003451   TA =003501                              MSG NO. 000002
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 C20100000000000 60400400000000000000  DCTRU       B


12.22.16.257        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000003
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0011
     001 630000001400200 30600000000120001000  CONREQ      C
     002 51C75D7ADB45018 24343535365555050030  T1223  E X    UW-4P
     003 0000000000001C2 00000000000000000702          GB
     004 00000000023840B 00000000000010702013         H'PK        #
     005 xxxxxxx6DB40011 xxxxxxxxxx5555000021  xxxxx     Q  M B [@
     006 xxxxxxxE1880037 xxxxxxxxxxxx42000067  xxxxxxx  &    16A  7
     007 000FF8FFFFFFFFF 00007770777777777777    ;';;;;;;    X____
     008 FFF3400001FFFFF 77771500000007777777  ;;M   G;;;  __4
     009 000000000000F6F 00000000000000007557          .          V.
     010 7C014034460D189 37000500150430150611  4 E MDXMFI  W@  D@Q


12.22.16.257        NETPUT  (006634)   HA =003451   TA =003501                              MSG NO. 000004
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 634000001400101 30640000000120000401  CONREQN     C@


12.22.16.352        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000005
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830700001000000 40603400000100000000  FCINIT


12.22.16.352        NETPUT  (006634)   HA =003451   TA =003501                              MSG NO. 000006
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 834700001000000 40643400000100000000  FCINITN      G


12.22.16.353        NETPUT  (006634)   HA =003451   TA =001614                              MSG NO. 000007
 ABT =02  ADR =0001  ABN =000001  ACT =04  STATUS = 00000000   TLC = 0050
     001 BD42094ED253B52 57241011235511235522  .THIS IS R  =B NRS5
     002 35676D55324E1ED 15263555252311160755  MV2 USING   #VVUS$AM
     003 45448DBED14E505 21242215575505162405  QTRM. ENTE  ED >QNP
     004 4AD4CF34550824E 22552317150524101116  R SOMETHIN  T-LSEP N
     005 1EF000000000000 07570000000000000000  G.           P
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 1 of 11)

```
1                                   RMV2     LOG FILE OUTPUT                                    83/06/16
                                    DATE RECORDED - 83/06/16                                    PAGE 00002
0

 12.22.16.771          NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000008
  ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
      001 830200001000040 40601000000100000100  FCACK

 12.23.18.412          NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012        MSG NO. 000009
  ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0047
      001 50816D385614B43 24100555160530245503  THE NEXT C  P M8V 4
      002 2014810D4152B49 10012201032405225511  HARACTER I  2 H T +I
      003 4ED06D553152982 23550155252305224602  S A USER-B  NPMU1R
      004 48504B99DB43201 22050113463555031001  REAK-2 CHA  $  9 42
      005 4810D4152BC0000 22010324052257000000  RACTER.     H T +@

 12.23.18.412          NETPUT   (006634)   HA =003451    TA =001614                             MSG NO. 000010
  ABT =01   ADR =0001  ABN =000002  ACT =04  STATUS = 00000000   TLC = 0050
      001 BD4205B4E15852D 57241005551605302455  .THE NEXT   =B 4AXR
      002 0C80520435054AD 03100122010324052255  CHARACTER   PH  CPT-
      003 253B41B554C54A6 11235501552523052246  IS A USER-  %;A5TEJ
      004 0921412E676D0C8 02220501134635550310  BREAK-2 CH  @   FVPH
      005 0520435054AF000 01220103240522570000  ARACTER.      CPT/

 12.23.18.413          NETPUT   (006634)   HA =003451    TA =001614                             MSG NO. 000011
  ABT =02   ADR =0001  ABN =000003  ACT =04  STATUS = 00000000   TLC = 0020
      001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
      002 679000000000000 31710000000000000000  Y?          &Y

 12.23.18.934          NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000012
  ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
      001 830200001000080 40601000000100000200  FCACK

 12.23.18.934          NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000013
  ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
      001 8302000010000C0 40601000000100000300  FCACK

 12.23.27.818          NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000014
  ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
      001 800004001000000 40000004000100000000  INTRUSR

 12.23.27.818          NETPUT   (006634)   HA =003451    TA =003501                             MSG NO. 000015
  ABT =03   ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 2 of 11)

```
     001 800100001000000 40000400000100000000  INTRRSP

12.23.27.818        NETPUT   (006634)   HA =003451    TA =003501                          MSG NO. 000016
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
     001 CB000000000000 62600000000000000000  ROMARK

12.23.27.818        NETPUT   (006634)   HA =003451    TA =001614                          MSG NO. 000017
 ABT =02  ADR =0001  ABN =000004  ACT =04  STATUS = 00000000   TLC = 0040
     001 BCE3ED0435093CE 57161755010324111716  .NO ACTION  <CM 5 <
     002 B5404B14EBED385 55240113051657551605   TAKEN. NE  KT 1N>S
     003 614B45394499E40 30245505162422317100  XT ENTRY?   AKE9D D
     004 000000000000000 00000000000000000000

12.23.27.827        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012       MSG NO. 000018
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
     001 CA0000353220202 62400000152310401002  BIMARK

12.23.28.833        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000019
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000000 40601000000100000000  FCACK

12.23.28.833        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000020
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000100 40601000000100000400  FCACK

12.23.47.074        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012       MSG NO. 000021
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0047
     001 50816D385614B43 24100555160530245503  THE NEXT C  P M8V 4
     002 2014810D4152B49 10012201032405225511  HARACTER I  2 H T +I
     003 4ED06D553152982 23550155252305224602  S A USER-B  NPMU1R
     004 48504B99CB43201 22050113463455031001  REAK-1 CHA  $  9 42
     005 4810D4152BC0000 22010324052257000000  RACTER.     H T +@

12.23.47.075        NETPUT   (006634)   HA =003451    TA =001614                          MSG NO. 000022
 ABT =01  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0050
     001 BD4205B4E15852D 57241005551605302455  .THE NEXT   =B 4AXR
     002 0C80520435054AD 03100122010324052255  CHARACTER   PH  CPT-
     003 253B41B554C54A6 11235501552523052246  IS A USER-  %;A5TEJ
     004 0921412E672C0C8 02220501135634550310  BREAK-1 CH  @    FRPH
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 3 of 11)

```
        005 0520435054AF000 01220103240522570000  ARACTER.      CPT/

 12.23.47.075          NETPUT  (006634)   HA =003451   TA =001614                          MSG NO. 000023
   ABT =02  ADR =0001  ABN =000006  ACT =04  STATUS = 00000000   TLC = 0020
        001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
        002 679000000000000 31710000000000000000  Y?          &Y

 12.23.48.087          NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063    MSG NO. 000024
   ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
        001 830200001000140 40601000000100000500  FCACK

 12.23.48.087          NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063    MSG NO. 000025
   ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
        001 830200001000180 40601000000100000600  FCACK

 12.24.06.067          NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063    MSG NO. 000026
   ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
        001 800003001000000 40000003000100000000  INTRUSR

 12.24.06.067          NETPUT  (006634)   HA =003451   TA =003501                          MSG NO. 000027
   ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
        001 800100001000000 40000400000100000000  INTRRSP

 12.24.06.067          NETPUT  (006634)   HA =003451   TA =003501                          MSG NO. 000028
   ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
        001 CB0000000000000 62600000000000000000  ROMARK

 12.24.06.067          NETPUT  (006634)   HA =003451   TA =001614                          MSG NO. 000029
   ABT =02  ADR =0001  ABN =000007  ACT =04  STATUS = 00000000   TLC = 0040
        001 BCE3ED0435093CE 57161755010324111716  .NO ACTION  <CM 5 <
        002 B5404B14EBED385 55240113051657551605   TAKEN. NE  KT 1N>S
        003 614B45394499E40 30245505162422317100  XT ENTRY?   AKE9D D
        004 000000000000000 00000000000000000000

 12.24.06.070          NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012    MSG NO. 000030
   ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
        001 CA0000000000000 62400000000000000000  BIMARK
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 4 of 11)

```
                         RMV2    LOG FILE OUTPUT                              83/06/16
                         DATE RECORDED - 83/06/16                             PAGE 00005

12.24.08.398        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000031
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000000 40601000000100000000  FCACK

12.24.08.421        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000032
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010001C0 40601000000100000700  FCACK

12.24.30.931        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012      MSG NO. 000033
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0036
     001 50816D385614B45 24100555160530245505  THE NEXT E  P M8V 4
     002 394499B494ED06D 16242231551123550155  NTRY IS A   S I INPM
     003 0921412ED24E109 02220501135511160411  BREAK INDI   !A.RN
     004 0C150F4AF000000 03012417225700000000  CATOR.       APT/

12.24.30.931        NETPUT   (006634)   HA =003451   TA =001614                            MSG NO. 000034
 ABT =01  ADR =0001  ABN =000008  ACT =04  STATUS = 00000000   TLC = 0040
     001 BD4205B4E15852D 57241005551605302455  .THE NEXT    =B 4AXR
     002 14E51266D253B41 05162422315511235501  ENTRY IS A  QNQ&M%;A
     003 B4248504BB49384 55022205011355111604   BREAK IND  4$  ;I8
     004 2430543D2BC0000 11030124172257000000  ICATOR.     BC CR<

12.24.30.932        NETPUT   (006634)   HA =003451   TA =001614                            MSG NO. 000035
 ABT =02  ADR =0001  ABN =000009  ACT =04  STATUS = 00000000   TLC = 0020
     001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
     002 679000000000000 31710000000000000000  Y?          &Y

12.24.31.984        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000036
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000200 40601000000100001000  FCACK

12.24.31.984        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000037
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000240 40601000000100001100  FCACK            $

12.24.33.521        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000038
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 800003001000000 40000003000100000000  INTRUSR
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 5 of 11)

```
12.24.33.521        NETPUT  (006634)   HA =003451   TA =003501                               MSG NO. 000039
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 800100001000000 40000400000100000000  INTRRSP

12.24.33.521        NETPUT  (006634)   HA =003451   TA =003501                               MSG NO. 000040
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
     001 CB0000000000000 62600000000000000000  ROMARK

12.24.33.522        NETPUT  (006634)   HA =003451   TA =001614                               MSG NO. 000041
 ABT =02  ADR =0001  ABN =000010  ACT =04  STATUS = 00000000   TLC = 0040
     001 BCE3ED0435093CE 57161755010324111716  .NO ACTION  <CM 5 <
     002 B5404B14EBED385 55240113051657551605   TAKEN. NE  KT 1N>S
     003 614B45394499E40 30245505162422317100  XT ENTRY?   AKE9D D
     004 000000000000000 00000000000000000000

12.24.33.525        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012          MSG NO. 000042
 ABT =03  ADR =0001  ABN =000000  ACT =02  STATUS = 00000000   TLC = 0002
     001 CA0000657300202 62400000312714001002  BIMARK

12.24.34.042        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063          MSG NO. 000043
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000000 40601000000100000000  FCACK

12.24.34.042        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063          MSG NO. 000044
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000280 40601000000100001200  FCACK              (

12.26.27.632        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012          MSG NO. 000045
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0003
     001 14E100000000000 05160400000000000000  END           A

12.26.27.632        NETPUT  (006634)   HA =003451   TA =001614                               MSG NO. 000046
 ABT =02  ADR =0001  ABN =000011  ACT =04  STATUS = 00000000   TLC = 0020
     001 BC73CF102645B46 57071717040231055506  .GOODBYE F  <SO &E4
     002 3D2B4E3D7BEF000 17225516172757570000  OR NOW..    CR4CW>P
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 6 of 11)

```
12.26.27.632        NETPUT  (006634)  HA =003451  TA =003501                        MSG NO. 000047
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
     001 C00000001000000 60000000000100000000  LSTOFF      @

12.26.27.632        NETPUT  (006634)  HA =003451  TA =003501                        MSG NO. 000048
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0002
     001 630600001000000 30603000000100000000  CONEND      C
     002 2411ADB6DB40000 11010655555555000000  IAF           A [M4

12.26.27.727        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063  MSG NO. 000049
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
     001 634600001000000 30643000000100000000  CONENDN     CF

12.26.41.158        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063  MSG NO. 000050
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0011
     001 630000001400200 30600000000120001000  CONREQ      C
     002 51C75D7ADB45018 24343535365555050030  T1223  E X    UW-4P
     003 0000000000001C2 00000000000000000702           GB
     004 00000000023840B 00000000000010702013          H'PK       #
     005 xxxxxxxxDB40011 xxxxxxxxxx5555000021  xxxxx    Q  M B [@
     006 xxxxxxxxx880037 xxxxxxxxxxxxxx000067  xxxxxxx  &    16A  7
     007 000FF8FFFFFFFFF 00007770777777777777    ;';;;;;;    X
     008 FFF3400001FFFFF 77771500000007777777  ;;M   G;;;    4
     009 000000000000F6F 00000000000000007557           .         V
     010 7C014034460D189 37000500150430150611  4 E MDXMFI  W@  D@Q

12.26.41.158        NETPUT  (006634)  HA =003451  TA =003501                        MSG NO. 000051
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
     001 634000001400101 30640000000120000401  CONREQN     C@

12.26.41.656        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063  MSG NO. 000052
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
     001 830700001000000 40603400000100000000  FCINIT

12.26.41.656        NETPUT  (006634)  HA =003451  TA =003501                        MSG NO. 000053
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
     001 834700001000000 40643400000100000000  FCINITN      G

12.26.41.656        NETPUT  (006634)  HA =003451  TA =001614                        MSG NO. 000054
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 7 of 11)

```
 ABT =02  ADR =0001  ABN =000001  ACT =04  STATUS = 00000000  TLC = 0050
    001 BD42094ED253B52 57241011235511235522  .THIS IS R  =B NRS5
    002 35676D55324E1ED 15263555252311160755  MV2 USING   #VVUS$AM
    003 45448DBED14E505 21242215575505162405  QTRM. ENTE  ED >QNP
    004 4AD4CF34550824E 22552317150524101116  R SOMETHIN  T-LSEP N
    005 1EF000000000000 07570000000000000000  G.           P

12.26.42.207        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000055
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000  TLC = 0001
    001 830200001000040 40601000000100000100  FCACK

12.27.27.901        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012        MSG NO. 000056
 ABT =01  ADR =0001  ABN =000000  ACT =04  STATUS = 00010000  TLC = 0100
    001 508253B494ED06D 24101123551123550155  THIS IS A   P S4 M
    002 51940504814112D 24312005011005010455  TYPEAHEAD   U @PH  -
    003 5054D4BAD14E505 24052324565505162405  TEST, ENTE  PTT QNP
    004 489387B414ED355 22111607550123551525  RING AS MU  T 8[ANSU
    005 0C8B5415852D053 03105524053024550123  CH TEXT AS    T  -
    006 B503D34C908C16D 55201723231102140555   POSSIBLE   ;P=4I AM
    007 50FB430554C5B4D 24175503012523055515  TO CAUSE M  P[C TE4
    008 54C50940C16D385 25142411201405551605  ULTIPLE NE  ULP   S
    009 5173D22ED08C3C3 24271722135502141703  TWORK BLOC  QSR.P <
    010 2D3B5540C24E16D 13235525201411160555  KS UPLINE   2S5T $AM

12.27.27.901        NETPUT  (006634)  HA =003451  TA =001614                                MSG NO. 000057
 ABT =01  ADR =0001  ABN =000002  ACT =04  STATUS = 00000000  TLC = 0110
    001 BD42094ED253B41 57241011235511235501  .THIS IS A  =B NRS4
    002 B54650141205044 55243120050110050104   TYPEAHEAD   TE A PD
    003 B5415352EB45394 55240523245655051624   TEST, ENT  5ASRKE9
    004 15224E1ED053B4D 05221116075501235515  ERING AS M  AR$AM ;M
    005 54322D505614B41 25031055240530245501  UCH TEXT A  T2-PV 4
    006 4ED40F4D3242305 23552017232311021405  S POSSIBLE   M@TS$#
    007 B543ED0C155316D 55241755030125230555   TO CAUSE   5CM  S
    008 355314250305B4E 15251424112014055516  MULTIPLE N  SU1BPO[N
    009 1545CF48BB4230F 05242717221355021417  ETWORK BLO   EOH;BO
    010 0CB4ED550309385 03132355252014111605  CKS UPLINE  PKNUPO
    011 000000000000000 00000000000000000000

12.27.27.902        NETPUT  (006634)  HA =003451  TA =001614                                MSG NO. 000058
 ABT =02  ADR =0001  ABN =000003  ACT =04  STATUS = 00000000  TLC = 0020
    001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
    002 679000000000000 31710000000000000000  Y?          &Y

12.27.52.164        NETGET  (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063        MSG NO. 000059
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 8 of 11)

```
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 830200001000080 4060100000010000200  FCACK


12.27.52.164         NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063       MSG NO. 000060
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 8302000010000C0 4060100000010000300  FCACK


12.27.52.169         NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012       MSG NO. 000061
 ABT =01  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0100
     001 50FB54205B4E154 24175524100555160524  TO THE NET  P[T [N
     002 5CF48BB41410309 27172213550120201411  WORK APPLI  EOH;AA
     003 0C15093CEB5048F 03012411171655202217  CATION PRO     <KPH
     004 1D204DBEDB54205 07220115575555241005  GRAM.  THE  QR [M5B
     005 B4939414E52D253 55111624051624551123   INTENT IS  4   E-%
     006 B543ED4C516D5C8 55241755230505552710   TO SEE WH  ;T>TE UH
     007 054B54205B5048F 01245524100555202217  AT THE PRO   KT [PH
     008 1D204DE13B51545 07220115702355212505  GRAM'S QUE  QR ^ 5 E
     009 54598804E10C24E 25054610011604141116  UE-HANDLIN  TY  A $
     010 1ED0CF105B5724C 07550317040555271114  G CODE WIL  AM Q 5RL


12.27.52.200         NETPUT   (006634)    HA =003451    TA =001614                          MSG NO. 000062
 ABT =01  ADR =0001  ABN =000004  ACT =04  STATUS = 00000000   TLC = 0110
     001 BD43ED50816D385 57241755241005551605  .TO THE NE  =CMP M8
     002 5173D22ED05040C 24271722135501202014  TWORK APPL  U ="M
     003 24305424F3AD412 11030124111716552022  ICATION PR  $OT$S-A
     004 3C748136FB6D508 17072201155755552410  OGRAM.  TH  #GH 06U
     005 16D24E505394B49 05551116240516245511  E INTENT I   RNPS 4
     006 4ED50FB53145B57 23552417552305055527  S TO SEE W   MP[S [W
     007 20152D50816D412 10012455241005552022  HAT THE PR    -P MA
     008 3C74813784ED455 17072201157023552125  OGRAM'S QU  #GH XNTU
     009 155166201384309 05250546100116041411  EUE-HANDLI   QF    0
     010 387B433C416D5C9 16075503170405552711  NG CODE WI    43D UI
     011 300000000000000 14000000000000000000  L            0


12.27.52.200         NETPUT   (006634)    HA =003451    TA =001614                          MSG NO. 000063
 ABT =02  ADR =0001  ABN =000005  ACT =04  STATUS = 00000000   TLC = 0020
     001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
     002 679000000000000 31710000000000000000  Y?          &Y


12.27.52.227         NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012       MSG NO. 000064
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0022
     001 32D10FB493AD508 14550417551116552410  L DO IN TH  2Q 4 -P
     002 253B49393501383 11235511162324011603  IS INSTANC   S4  P
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 9 of 11)

```
    003 16F000000000000 05570000000000000000  E.          P

12.27.52.674        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000065
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
    001 830200001000100 40601000000100000400  FCACK

12.27.52.674        NETPUT   (006634)  HA =003451   TA =001614                          MSG NO. 000066
 ABT =01  ADR =0001  ABN =000006  ACT =04  STATUS = 00000000   TLC = 0030
    001 BCCB443ED24EB54 57145504175511165524  .L DO IN T  <KD>RN5
    002 2094ED24E4D404E 10112355111623240116  HIS INSTAN  B NRNM@N
    003 0C5BC0000000000 03055700000000000000  CE.         [@

12.27.53.777        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000067
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
    001 830200001000140 40601000000100000500  FCACK

12.27.53.777        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000068
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
    001 830200001000180 40601000000100000600  FCACK

12.27.53.778        NETPUT   (006634)  HA =003451   TA =001614                          MSG NO. 000069
 ABT =02  ADR =0001  ABN =000007  ACT =04  STATUS = 00000000   TLC = 0020
    001 BCE15852D14E512 57160530245505162422  .NEXT ENTR  <AXRQNQ
    002 679000000000000 31710000000000000000  Y?          &Y

12.27.54.760        NETGET   (006312)  ACN =0000  HA =003451  TA =003501  TLMAX =0063      MSG NO. 000070
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
    001 8302000010001C0 40601000000100000700  FCACK

12.28.07.750        NETGETL (006326)  ALN =0001  HA =003451  TA =001602  TLMAX =0012      MSG NO. 000071
 ABT =02  ADR =0001  ABN =000000  ACT =04  STATUS = 00000000   TLC = 0008
    001 4C855410F5CE000 23102524041727160000  SHUTDOWN    L T UN

12.28.07.751        NETPUT   (006634)  HA =003451   TA =001614                          MSG NO. 000072
 ABT =02  ADR =0001  ABN =000008  ACT =04  STATUS = 00000000   TLC = 0020
    001 BC2645B463D2156 57023105550617220526  .BYE FOREV  <&E4CR
    002 152D80000000000 05226600000000000000  ER!         ARX
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 10 of 11)

```
                         RMV2    LOG FILE OUTPUT                                  83/06/16
                         DATE RECORDED - 83/06/16                                 PAGE 00011


12.28.07.751        NETPUT   (006634)   HA =003451   TA =001614                   MSG NO. 000073
 ABT =02  ADR =0001  ABN =000009  ACT =04  STATUS = 00000000   TLC = 0020
     001 BD32155043D73AD 57231025240417271655  .SHUTDOWN    =2 PCW
     002 0CF349387000000 03171511160700000000  COMING       P04

12.28.07.751        NETPUT   (006634)   HA =003451   TA =003501                   MSG NO. 000074
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0001
     001 C00000001000000 60000000000100000000  LSTOFF       @

12.28.07.751        NETPUT   (006634)   HA =003451   TA =003501                   MSG NO. 000075
 ABT =03  ADR =0000  ABN =000000  ACT =01  STATUS = 00000000   TLC = 0002
     001 630600001000000 30603000000100000000  CONEND       C
     002 2411ADB6DB40000 11010655555555000000  IAF           A [M4

12.28.08.750        NETOFF   (003500)       DATE =83/06/16                         MSG NO. 000076
```

Figure 8-11. Debug Log File Listing for ECHO-RMV2 (Sheet 11 of 11)

```
NAM STATISTICS GATHERING STARTED
NETON   DATE  83/06/16.   TIME 12.21.41.

NAM STATISTICS GATHERING TERMINATED
NETOFF  DATE  83/06/16.   TIME 12.28.09.

CPU TIME USED:           0.030 SEC


NUMBER OF PROCEDURE CALLS
    NETGET                  67
    NETGETL                 39
    NETPUT                  35
    NETWAIT                 27

NUMBER OF WORKLIST TRANSFER ATTEMPTS
    SUCCESSFUL              73

NUMBER OF INPUT/OUTPUT BLOCKS TRANSFERRED
    INPUT   ABT=0           56
    INPUT   ABT=1            2
    INPUT   ABT=2            6
    INPUT   ABT=3           31
    OUTPUT  ABT=1            6
    OUTPUT  ABT=2           14
    OUTPUT  ABT=3           15

NUMBER OF ERRORS
```

Figure 8-12. Statistics File Listing for ECHO-RMV-2

```
THIS IS RMV2 USING QTRM. ENTER SOMETHING.
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?
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 entry is a break indicator.
THE NEXT ENTRY IS A BREAK INDICATOR.
NEXT ENTRY?

NO ACTION TAKEN. NEXT ENTRY?
end
GOODBYE FOR NOW..
RMV2    CONNECT TIME 00.04.11.
JSN: ABEF, NAMIAF
/bye,rmv2
UN=xxxxxxx  LOG OFF  12.26.38.
JSN=ABEF     SRU-S       2.007
IAF      CONNECT TIME 00.00.10.
THIS IS RMV2 USING QTRM. ENTER SOMETHING.
THis is a typeahead test, entering as much text as possible to cause multiple
network blocks upline to the network application program.  The intent is to see
what the program's queue-handling code will do in this instance.
THIS IS A TYPEAHEAD TEST, ENTERING AS MUCH TEXT AS POSSIBLE TO CAUSE MULTIPLE
NETWORK BLOCKS UPLINE
NEXT ENTRY?
TO THE NETWORK APPLICATION PROGRAM.  THE INTENT IS TO SEE WHAT THE PROGRAM'S
QUEUE-HANDLING CODE WIL
NEXT ENTRY?
L DO IN THIS INSTANCE.
NEXT ENTRY?
shutdown
BYE FOREVER!
SHUTDOWN COMING
RMV2    CONNECT TIME 00.01.27.
JSN: ABEH, NAMIAF
```

Figure 8-13. ECHO-RMV2 Sample Dialog

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 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.

Translation of 7-bit or 12-bit ASCII to 6-bit display code causes character folding from the 128-character ASCII set to the 63- or 64-character ASCII subset, with the special character substitutions shown in figure A-1.

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.

		63- or 64-Character Subset		
12-Bit ASCII (Octal)		6-Bit Display Code (Octal)		12-Bit ASCII (Octal)
0140 (`)		74 (@)		0100 (@)
0173 ({)		61 ([)		0133 ([)
0174 (|)	Input →	75 (\)	Output →	0134 (\)
0175 (})		62 (])		0135 (])
0176 (~)		76 (^)		0136 (^)

Figure A-1. ASCII Character Folding

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 character 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.

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 paper tape support is requested, the device control 3 code also is deleted. Some of these code deletions can be suppressed by using the full-ASCII and special 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 character 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 or I 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	Low Cost System Print Band
CDC graphic 64-character set	596-1	--
ASCII graphic 64-character set	596-5	530-1
ASCII graphic 95-character set	596-6	530-2

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 column of the repeated line following 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

This section describes the types of network failure that are possible. Each type of failure has 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 consoles.

If the Network Access Method fails (specifically, if NIP communication fails), the network software dumps NAM's field length to a special file 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, consoles (and to some other types of devices) informing the terminal that the host is unavailable. After recovery, all logical links are reinitialized and new connections are made.

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 CCP tables called 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 CS 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 CS.

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 character mode transmission. Character 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-2 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-2. 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
@ commercial at	@ commercial at	74††	7401††	0100
\ reverse slant	\ reverse slant	75	75	0134
^ circumflex		76		
; semicolon	; semicolon	77	77	0073
	^ circumflex	76††	7402	0136
	: colon††	74††	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
	{ opening brace	61††	7633	0173
	| vertical line	75††	7634	0174
	} closing brace	62††	7635	0175
	~ tilde	76††	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.

††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 (columns 0–7)

96-Character Subset (columns 2–7)

Graphic 64-Character Subset (columns 2–5)

Bits												
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
b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
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 or 11-8-2§	12-8-7 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 or 12-8-2§	12-8-4 or 12-0§
> greater than	> greater than	> greater than	73	73	0076	11-8-7	0-8-6
≤ less/equal	@ commercial at	@ commercial at	74††	7401††	0100	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
		^ circumflex	76††	7402	0136		
		: colon††		7404††	0072		
		` grave accent	74††	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	61††	7633	0173		
		| vertical line	75††	7634	0174		
		} clos. brace	62††	7635	0175		
		~ tilde	76††	7636	0176		

†Available only on NOS.

††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	ǀ 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	⑂
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	⑂
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 AND LINE PRINTERS IN TERMINAL CLASSES 9, 14, 16, 17, AND 18 (HASP, HPRE, 2780, 3270, AND 3780)

Terminal EBCDIC				Network ASCII (Normalized Mode Use)			
Hex. Code	Octal Code	Graphic†	Control Character††	Hex. Code†††	Octal Code†††	Graphic	Control Character††
00	000		NUL	00	000		null
01	001		SOH	01	001		start of header
02	002		STX	02	002		start of text
03	003		ETX	03	003		end of text
04	004		PF	20	040	space	
05	005		HT	09	011		horizontal tabulate
06	006		LC	20	040	space	
07	007		DEL	7F	177		delete
08	010		undefined	20	040	space	
09	011		undefined	20	040	space	
0A	012		SMM	20	040	space	
0B	013		VT	0B	013		vertical tabulate
0C	014		FF	0C	014		form feed
0D	015		CR	0D	015		carriage return
0E	016		SO	0E	016		shift out
0F	017		SI	0F	017		shift in
10	020		DLE	10	020		data link escape
11	021		DC1	11	021		device control 1
12	022		DC2	12	022		device control 2
13	023		TM	13	023		device control 3
14	024		RES	20	040	space	
15	025		NL	20	040	space	
16	026		BS	08	010		backspace
17	027		IL	20	040	space	
18	030		CAN	18	030		cancel
19	031		EM	19	031		end of medium
1A	032		CC	20	040	space	
1B	033		CU1	20	040	space	
1C	034		IFS	1C	034		file separator
1D	035		IGS	1D	035		group separator
1E	036		IRS	1E	036		record separator
1F	037		IUS	1F	037		unit separator
20	040		DS	20	040	space	
21	041		SOS	20	040	space	
22	042		FS	20	040	space	
23	043		undefined	20	040	space	
24	044		BYP	20	040	space	
25	045		LF	0A	012		linefeed
26	046		ETB or EOB	17	027		end of transmission block
27	047		ESC or PRE	1B	033		escape
28	050		undefined	20	040	space	
29	051		undefined	20	040	space	
2A	052		SM	20	040	space	
2B	053		CU2	20	040	space	
2C	054		undefined	20	040	space	
2D	055		ENQ	05	005		enquiry
2E	056		ACK	06	006		positive acknowledgment
2F	057		BEL	07	007		bell
30	060		undefined	20	040	space	
31	061		undefined	20	040	space	
32	062		SYN	16	026		synchronous idle
33	063		undefined	20	040	space	
34	064		PN	20	040	space	
35	065		RS	20	040	space	
36	066		UC	20	040	space	
37	067		EOT	04	004		end of transmission
38	070		undefined	20	040	space	
39	071		undefined	20	040	space	
3A	072		undefined	20	040	space	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 9, 14, 16, 17, AND 18 (HASP, HPRE, 2780, 3270, AND 3780) (Contd)

Terminal EBCDIC				Network ASCII (Normalized Mode Use)			
Hex. Code	Octal Code	Graphic†	Control Character††	Hex. Code†††	Octal Code†††	Graphic	Control Character††
3B	073		CU3	20	040	space	
3C	074		DC4	14	024		device control 4
3D	075		NAK	15	025		negative acknowledgement
3E	076		undefined	20	040	space	
3F	077		SUB	1A	032		substitute
40	100	space		20	040	space	
41 thru 49	101 thru 111		undefined	20	040	space	
4A	112	¢		5B	133	[	
4B	113	.		2E	056	.	
4C	114	<		3C	074	<	
4D	115	(		28	050	(	
4E	116	+		2B	053	+	
4F	117	|		21	041	!	
50	120	&		26	046	&	
51 thru 59	121 thru 131		undefined	20	040	space	
5A	132	!		5D	135	]	
5B	133	$		24	044	$	
5C	134	*		2A	052	*	
5D	135	)		29	051	)	
5E	136	;		3B	073	;	
5F	137	¬		5E	136	^	
60	140	-		2D	055	-	
61	141	/		2F	057	/	
62 thru 69	142 thru 151		undefined	20	040	space	
6A	152	¦		7C	174	¦	
6B	153	,		2C	054	,	
6C	154	%		25	045	%	
6D	155	_		5F	137	_	
6E	156	>		3E	076	>	
6F	157	?		3F	077	?	
70 thru 78	160 thru 170		undefined	20	040	space	
79	171	`		60	140	`	
7A	172	:		7A	172	:	
7B	173	#		23	043	#	
7C	174	@		40	100	@	
7D	175	'		27	047	'	
7E	176	=		3D	075	=	
7F	177	"		22	042	"	
80	200		undefined	20	040	space	
81	201	a		61	141	a	
82	202	b		62	142	b	
83	203	c		63	143	c	
84	204	d		64	144	d	
85	205	e		65	145	e	
86	206	f		66	146	f	
87	207	g		67	147	g	
88	210	h		68	150	h	
89	211	i		69	151	i	
8A thru 90	212 thru 220		undefined	20	040	space	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 9, 14, 16, 17, AND 18 (HASP, HPRE, 2780, 3270, AND 3780) (Contd)

Terminal EBCDIC				Network ASCII (Normalized Mode Use)			
Hex. Code	Octal Code	Graphic†	Control Character††	Hex. Code†††	Octal Code†††	Graphic	Control Character††
91	221	j		6A	152	j	
92	222	k		6B	153	k	
93	223	l		6C	154	l	
94	224	m		6D	155	m	
95	225	n		6E	156	n	
96	226	o		6F	157	o	
97	227	p		70	160	p	
98	230	q		71	161	q	
99	231	r		72	162	r	
9A thru A0	232 thru 240		undefined	20	040	space	
A1	241	~		7E	176	~	
A2	242	s		73	163	s	
A3	243	t		74	164	t	
A4	244	u		75	165	u	
A5	245	v		76	166	v	
A6	246	w		77	167	w	
A7	247	x		78	170	x	
A8	250	y		79	171	y	
A9	251	z		7A	172	z	
AA thru BF	252 thru 277		undefined	20	040	space	
C0	300	{		7B	173	{	
C1	301	A		41	101	A	
C2	302	B		42	102	B	
C3	303	C		43	103	C	
C4	304	D		44	104	D	
C5	305	E		45	105	E	
C6	306	F		46	106	F	
C7	307	G		47	107	G	
C8	310	H		48	110	H	
C9	311	I		49	111	I	
CA	312		undefined	20	040	space	
CB	313		undefined	20	040	space	
CC	314	⑀		20	040	space	
CD	315		undefined	20	040	space	
CE	316	⑂		20	040	space	
CF	317		undefined	20	040	space	
D0	320	}		7E	175	}	
D1	321	J		4A	112	J	
D2	322	K		4B	113	K	
D3	323	L		4C	114	L	
D4	324	M		4D	115	M	
D5	325	N		4E	116	N	
D6	326	O		4F	117	O	
D7	327	P		50	120	P	
D8	330	Q		51	121	Q	
D9	331	R		52	122	R	
DA thru DF	332 thru 337		undefined	20	040	space	
E0	340	\		5C	134	\	
E1	341		undefined	20	040	space	
E2	342	S		53	123	S	
E3	343	T		54	124	T	
E4	344	U		55	125	U	
E5	345	V		56	126	V	

TABLE A-8. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 9, 14, 16, 17, AND 18 (HASP, HPRE, 2780, 3270, AND 3780) (Contd)

Terminal EBCDIC				Network ASCII (Normalized Mode Use)			
Hex. Code	Octal Code	Graphic†	Control Character††	Hex. Code†††	Octal Code†††	Graphic	Control Character††
E6	346	W		57	127	W	
E7	347	X		58	130	X	
E8	350	Y		59	131	Y	
E9	351	Z		5A	132	Z	
EA	352		undefined	20	040	space	
EB	353		undefined	20	040	space	
EC	354	⑁		20	040	space	
ED	355		undefined	20	040	space	
thru	thru						
EF	357						
F0	360	0		30	060	0	
F1	361	1		31	061	1	
F2	362	2		32	062	2	
F3	363	3		33	063	3	
F4	364	4		34	064	4	
F5	365	5		35	065	5	
F6	366	6		36	066	6	
F7	367	7		37	067	7	
F8	370	8		38	070	8	
F9	371	9		39	071	9	
FA	372	|		20	040	space	
FB	373		undefined	20	040	space	
thru	thru						
FF	377						

†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.

††Not used for output to line printers. Translation to a space (100 octal) occurs.

†††Shown with zero parity (eighth or uppermost bit is always zero).

TABLE A-9. CHARACTER CODE TRANSLATIONS, ASCII CHARACTER SET CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
00	000		NUL or @	00	000		null
03	003	▲	ETX or C	03	003		end of text
05	005		ENQ or WRU or E	05	005		enquiry
06	006		ACK or RU or F	06	006		positive acknowledgement
09	011		HT or I	09	011		horizontal tabulate
0A	012		LF or NL or ↓ or J	0A	012		linefeed
0C	014		FF or FORM or L	0C	014		formfeed
0F	017	≥	SI or O	0F	017		shift in
11	021		DC1 or X-ON or Q	11	021		device control 1
12	022		DC2 or TAPE or R	12	022		device control 2
14	024		DC4 or TAPE or T	14	024		device control 4
17	027		ETB or W	17	027		end transmission block
18	030		CAN or CLEAR or X	18	030		cancel
1B	033		ESC or ESCAPE or [	1B	033		escape
1D	035		GS or]	1D	035		group separator
1E	036		RS or ^	1E	036		record separator
21	041	!		21	041	!	
22	042	"		22	042	"	
24	044	$		24	044	$	
27	047	'		27	047	'	
28	050	(		28	050	(	
2B	053	+		2B	053	+	
2D	055	-		2D	055	-	
2E	056	.		2E	056	.	
30	060	0		30	060	0	
33	063	3		33	063	3	
35	065	5		35	065	5	
36	066	6		36	066	6	
39	071	9		39	071	9	
3A	072	:		3A	072	:	
3C	074	<		3C	074	<	
3F	077	?		3F	077	?	
41	101	A		41	101	A	
42	102	B		42	102	B	
44	104	D		44	104	D	
47	107	G		47	107	G	
48	110	H		48	110	H	
4B	113	K		4B	113	K	
4D	115	M		4D	115	M	
4E	116	N		4E	116	N	
50	120	P		50	120	P	
53	123	S		53	123	S	
55	125	U		55	125	U	
56	126	V		56	126	V	
59	131	Y		59	131	Y	
5A	132	Z		5A	132	Z	
5C	134	\		5C	134	\	
5F	137	or ←		5F	137		
60	140	`		60	140	`	
63	143	c		63	143	c	
65	145	e		65	145	e	
66	146	f		66	146	f	
69	151	i		69	151	i	
6A	152	j		6A	152	j	
6C	154	l		6C	154	l	
6F	157	o		6F	157	o	
71	161	q		71	161	q	
72	162	r		72	162	r	

TABLE A-9. CHARACTER CODE TRANSLATIONS, ASCII CHARACTER SET CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
74	164	t		74	164	t	
77	167	w		77	167	w	
78	170	x		78	170	x	
7B	173	{		7B	173	{	
7C	174	¦ or ↑ or |		7C	174	¦	
7D	175	}		7D	175	}	
7E	176	~ or ¬		7E	176	~	
81	201		SOH or A	01	001		start of header
82	202		STX or B	02	002		start of text
84	204		EOT or D	04	004		end of transmission
87	207		BELL or G	07	007		bell
88	210		BS or ← or H	08	010		backspace
8B	213		VT or K	0B	013		vertical tabulate
8D	215		CR or RETURN or M	0D	015		carriage return
8E	216	≪	SO or N	0E	016		shift out
90	220		DLE or P	10	020		data link escape
93	223		DC3 or X-OFF or S	13	023		device control 3
95	225		NAK or → or U	15	025		negative acknowledgement
96	226		SYN or LINE CLEAR or V	16	026		synchronous idle
99	231		EM or RESET or Y	19	031		end of medium
9A	232		SUB or ↑ or Z	1A	032		substitute
9C	234		FS or \	1C	034		file separator
9F	237		US or −	1F	037		unit separator
A0	240	SPACE or blank		20	040	space	
A3	243	#		23	043	#	
A5	245	%		25	045	%	
A6	246	&		26	046	&	
A9	251	)		29	051	)	
AA	252	*		2A	052	*	
AC	254	,		2C	054	,	
AF	257	/		2F	057	/	
B1	261	1		31	061	1	
B2	262	2		32	062	2	
B4	264	4		34	064	4	
B7	267	7		37	067	7	
B8	270	8		38	070	8	
BB	273	;		3B	073	;	
BD	275	=		3D	075	=	
BE	276	>		3E	076	>	
C0	300	@		40	100	@	
C3	303	C		43	103	C	
C5	305	E		45	105	E	
C6	306	F		46	106	F	
C9	311	I		49	111	I	
CA	312	J		4A	112	J	
CC	314	L		4C	114	L	
CF	317	O		4F	117	O	
D1	321	Q		51	121	Q	
D2	322	R		52	122	R	
D4	324	T		54	124	T	
D7	327	W		57	127	W	
D8	330	X		58	130	X	
DB	333	[		5B	133	[	
DD	335	]		5D	135	]	
DE	336	∧ or ¬		5E	136	∧	
E1	341	a		61	141	a	

TABLE A-9. CHARACTER CODE TRANSLATIONS, ASCII CHARACTER SET CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
E2	342	b		62	142	b	
E4	344	d		64	144	d	
E7	347	g		67	147	g	
E8	350	h		68	150	h	
EB	353	k		6B	153	k	
ED	355	m		6D	155	m	
EE	356	n		6E	156	n	
F0	360	p		70	160	p	
F3	363	s		73	163	s	
F5	365	u		75	165	u	
F6	366	v		76	166	v	
F9	371	y		79	171	y	
FA	372	z		7A	172	z	
FF	377	■	DEL or RUBOUT	7F	177		delete

†Shown with even parity, which is the default for these terminal classes (unless PA=N or PA=I, 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-10. CHARACTER CODE TRANSLATIONS, APL TYPEWRITER-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
74	164	T		54	124	T	
77	167	W		57	127	W	
78	170	X		58	130	X	
7B	173	{		7B	173	{	
7C	174	⊣		6B	153	⊣	
7D	175	}		7D	175	}	
7E	176	$		24	044	$	
81	201		SOH or Ⓐ	01	001		start of header
82	202		STX or Ⓑ	02	002		start of text
84	204		EOT or Ⓓ	04	004		end of transmission
87	207		BELL or Ⓖ	07	007		bell
88	210		BS or ← or Ⓗ	08	010		backspace
8B	213		VT or Ⓚ	0B	013		vertical tabulate
8D	215		CR or RETURN or Ⓜ	0D	015		carriage return
8E	216	≪	SO or Ⓝ	0E	016	≪	shift out
90	220		DLE or Ⓟ	10	020		data link escape
93	223		DC3 or X-OFF or Ⓢ	13	023		device control 3
95	225		NAK or → or Ⓤ	15	025		negative acknowledgement
96	226		SYN or LINE CLEAR or Ⓥ	16	026		synchronous idle
99	231		EM or RESET or Ⓨ	19	031		end of medium
9A	232		SUB or ↑ or Ⓩ	1A	032		substitute
9C	234		FS or ⃠	1C	034		file separator
9F	237		US or ⊖	1F	037		unit separator
A0	240	SPACE or blank		20	040	space	
A3	243	<		3C	074	<	
A5	245	=		3D	075	=	
A6	246	>		3E	076	>	
A9	251	∧		26	046	∧	
AA	252	≠		22	042	≠	
AC	254	,		2C	054	,	
AF	257	/		2F	057	/	
B1	261	1		31	061	1	
B2	262	2		32	062	2	
B4	264	4		34	064	4	
B7	267	7		37	067	7	
B8	270	8		38	070	8	
BB	273	[		5B	133	[	
BD	275	×		66	146	×	
BE	276	:		3A	072	:	
C0	300	¯		5E	136	¯	
C3	303	∩		63	143	∩	
C5	305	∈		65	145	∈	
C6	306			5F	137		
C9	311	⍳		69	151	⍳	
CA	312	∘		6A	152	∘	
CC	314	⎕		6C	154	⎕	
CF	317	○		6F	157	○	
D1	321	?		3F	077	?	
D2	322	ρ		72	162	ρ	
D4	324	~		74	164	~	
D7	327	ω		77	167	ω	
D8	330	⊃		78	170	⊃	
DB	333	←		70	160	←	
DD	335	→		71	161	→	
DE	336	≥		7C	174	≥	
E1	341	A̲		41	101	A̲	

TABLE A-10. CHARACTER CODE TRANSLATIONS, APL TYPEWRITER-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
00	000		NUL or @	00	000		null
03	003	▲	ETX or C	03	003		end of text
05	005		ENQ or WRU or E	05	005		enquiry
06	006		ACK or RU or F	06	006		positive acknowledgement
09	011		HT or I	09	011		horizontal tabulate
0A	012		LF or NL or ↓ or J	0A	012		linefeed
0C	014		FF or FORM or L	0C	014		formfeed
0F	017	≥	SI or O	0F	017		shift in
11	021		DC1 or X-ON or Q	11	021		device control 1
12	022		DC2 or TAPE or R	12	022		device control 2
14	024		DC4 or TAPE or T	14	024		device control 4
17	027		ETB or W	17	027		end transmission block
18	030		CAN or CLEAR or X	18	030		cancel
1B	033		ESC or ESCAPE or [	1B	033		escape
1D	035		GS or]	1D	035		group separator
1E	036		RS or ∧	1E	036		record separator
21	041	•		23	043	•	
22	042	;		29	052	;	
24	044	≤		40	100	≤	
27	047	⌶		5D	135	⌶	
28	050	∨		21	041	∨	
2B	053	+		25	045	+	
2D	055	÷		2B	053	÷	
2E	056	.		2E	056	.	
30	060	0		30	060	0	
33	063	3		33	063	3	
35	065	5		35	065	5	
36	066	6		36	066	6	
39	071	9		39	071	9	
3A	072	(		28	050	(	
3C	074	;		3B	073	;	
3F	077	\		5C	134	\	
41	101	∝		61	141	∝	
42	102	⊥		62	142	⊥	
44	104	⌊		64	144	⌊	
47	107	∇		67	147	∇	
48	110	∆		68	150	∆	
4B	113	`		27	047	`	
4D	115	|		6D	155	|	
4E	116	⊤		6E	156	⊤	
50	120	*		2A	052	*	
53	123	⌈		73	163	⌈	
55	125	↓		75	165	↓	
56	126	∪		76	166	∪	
59	131	↑		79	171	↑	
5A	132	⊂		7A	172	⊂	
5C	134	⊢		7E	176	⊢	
5F	137	¯		2D	055	¯	
60	140	◊		60	140	◊	
63	143	C		43	103	C	
65	145	E		45	105	E	
66	146	F		46	106	F	
69	151	I		47	111	I	
6A	152	J		4A	112	J	
6C	154	L		4C	114	L	
6F	157	O		4F	117	O	
71	161	Q		51	121	Q	
72	162	R		52	122	R	

TABLE A-10. CHARACTER CODE TRANSLATIONS, APL TYPEWRITER-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 713, X3.64, H2000, T4014, M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
E2	342	B		42	102	B	
E4	344	D		44	104	D	
E7	347	G		47	107	G	
E8	350	H		48	110	H	
EB	353	K		4B	113	K	
ED	355	M		4D	115	M	
EE	356	N		4E	116	N	
F0	360	P		50	120	P	
F3	363	S		53	123	S	
F5	365	U		55	125	U	
F6	366	V		56	126	V	
F9	371	Y		59	131	Y	
FA	372	Z		5A	132	Z	
FF	377	■	DEL or RUBOUT	7F	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-11. CHARACTER CODE TRANSLATIONS, APL BIT-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 753, 751, H2000, T4014, AND M40)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
00	000		NUL or @	00	000		null
03	003	▲	ETX or C	03	003		end of text
05	005		ENQ or WRU or E	05	005		enquiry
06	006		ACK or RU or F	06	006		positive acknowledgement
09	011		HT or I	09	011		horizontal tabulate
0A	012		LF or NL or ↓ or J	0A	012		linefeed
0C	014		FF or FORM or L	0C	014		formfeed
0F	017	≫	SI or O	0F	017		shift in
11	021		DC1 or X-ON or Q	11	021		device control 1
12	022		DC2 or TAPE or R	12	022		device control 2
14	024		DC4 or TAPE or T	14	024		device control 4
17	027		ETB or W	17	027		end transmission block
18	030		CAN or CLEAR or X	18	030		cancel
1B	033		ESC or ESCAPE or [	1B	033		escape
1D	035		GS or]	1D	035		group separator
1E	036		RS or ∧	1E	036		record separator
21	041	¨		23	043	¨	
22	042	¯		5E	136	¯	
24	044	≤		40	100	≤	
27	047	≥		3E	076	≥	
28	050	≠		22	042	≠	
2B	053	(		28	050	(	
2D	055	+		2B	053	+	
2E	056	.		2E	056	.	
30	060	0		30	060	0	
33	063	3		33	063	3	
35	065	5		35	065	5	
36	066	6		36	066	6	
39	071	9		39	071	9	
3A	072	]		5D	135	]	
3C	074	;		3B	073	;	
3F	077	\		5C	134	\	
41	101	⍺		61	141	⍺	
42	102	⊥		62	142	⊥	
44	104	⌊		64	144	⌊	
47	107	∇		67	147	∇	
48	110	∆		68	150	∆	
4B	113	`		27	047	`	
4D	115	∣		6D	155	∣	
4E	116	⊤		6E	156	⊤	
50	120	*		2A	052	*	
53	123	⌈		73	163	⌈	
55	125	↓		75	165	↓	
56	126	∪		76	166	∪	
59	131	↑		79	171	↑	
5A	132	⊂		7A	172	⊂	
5C	134	◇		60	140	◇	
5F	137	∧		26	046	∧	
60	140	→		71	161	→	
63	143	C		43	103	C	
65	145	E		45	105	E	
66	146	F		46	106	F	
69	151	I		49	111	I	
6A	152	J		4A	112	J	
6C	154	L		4C	114	L	
6F	157	O		4F	117	O	
71	161	Q		51	121	Q	
72	162	R		52	122	R	

TABLE A-11. CHARACTER CODE TRANSLATIONS, APL BIT-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 753, 751, H2000, T4014, AND M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
74	164	T		54	124	T	
77	167	W		57	127	W	
78	170	X		58	130	X	
7B	173	⊣		6B	153	⊣	
7C	174	$		24	044	$	
7D	175	}		7D	160	}	
7E	176	÷		25	045	÷	
81	201		SOH or A	01	001		start of header
82	202		STX or B	02	002		start of text
84	204		EOT or D	04	004		end of transmission
87	207		BELL or G	07	007		bell
88	210		BS or ← or H	08	010		backspace
8B	213		VT or K	0B	013		vertical tabulate
8D	215		CR or RETURN or M	0D	015		carriage return
8E	216	≪	SO or N	0E	016		shift out
90	220		DLE or P	10	020		data link escape
93	223		DC3 or X-OFF or S	13	023		device control 3
95	225		NAK or → or U	15	025		negative acknowledgement
96	226		SYN or LINE CLEAR or V	16	026		synchronous idle
99	231		EM or RESET or Y	19	031		end of medium
9A	232		SUB or ↑ or Z	1A	032		substitute
9C	234		FS or \	1C	034		file separator
9F	237		US or -	1F	037		unit separator
A0	240	SPACE or blank		20	040	space	
A3	243	<		3C	074	<	
A5	245	=		3D	075	=	
A6	246	≥		7C	174	≥	
A9	251	∨		21	041	∨	
AA	252	)		29	051	)	
AC	254	,		2C	054	,	
AF	257	/		2F	057	/	
B1	261	1		31	061	1	
B2	262	2		32	062	2	
B4	264	4		34	064	4	
B7	267	7		37	067	7	
B8	270	8		38	070	8	
BB	273	[		5B	133	[	
BD	275	-		2D	055	-	
BE	276	:		3A	072	:	
C0	300	←		70	160	←	
C3	303	∩		63	143	∩	
C5	305	∊		65	145	∊	
C6	306			5F	137		
C9	311	⍳		69	151	⍳	
CA	312	∘		6A	152	∘	
CC	314	⎕		6C	154	⎕	
CF	317	○		6F	157	○	
D1	321	?		3F	077	?	
D2	322	⍴		72	162	⍴	
D4	324	~		74	164	~	
D7	327	⍵		77	167	⍵	
D8	330	⊃		78	170	⊃	
DB	333	⊢		7E	176	⊢	
DD	335	{		7B	173	{	
DE	336	X		66	146	X	
E1	341	A		41	101	A	

TABLE A-11. CHARACTER CODE TRANSLATIONS, APL BIT-PAIRING CONSOLES IN TERMINAL CLASSES 1, 2, AND 5 THROUGH 8 (M33, 753, 751, H2000, T4014, AND M40) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	ASCII-APL Graphic	Control Character††	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
E2	342	B		42	102	B	
E4	344	D		44	104	D	
E7	347	G		47	107	G	
E8	350	H		48	110	H	
EB	353	K		4B	113	K	
ED	355	M		4D	115	M	
EE	356	N		4E	116	N	
F0	360	P		50	120	P	
F3	363	S		53	123	S	
F5	365	U		55	125	U	
F6	366	V		56	126	V	
F9	371	Y		59	131	Y	
FA	372	Z		5A	132	Z	
FF	377	■	DEL or RUBOUT	7F	177		delete

†Shown with even parity, which is the default for these terminal classes (unless PA=N or PA=I, 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-12. CHARACTER CODE TRANSLATIONS, ASCII CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 10 AND 15 (200UT AND 734)

Terminal ASCII†				Network ASCII (Normalized Mode Use)				
		Keyboard or Printer Graphic		Input or Output		Console Output Only		
Hex. Code††	Octal Code††	ASCII	CDC	Hex. Code†††	Octal Code†††	Hex. Code†††	Octal Code†††	Graphic
20	040	blank	blank	20	040			space
23	043	#		23	043			#
25	045	%	%	25	045			%
26	046	&		26	046			&
29	051	)	)	29	051			)
2A	052	*	*	2A	052			*
2C	054	,	,	2C	054			,
2F	057	/	/	2F	057			/
31	061	1	1	31	061			1
32	062	2	2	32	062			2
34	064	4	4	34	064			4
37	067	7	7	37	067			7
38	070	8	8	38	070			8
3B	073	;	;	3B	073			;
3D	075	=	=	3D	075			=
3E	076	>	>	3E	076			>
40	100	@	≤	40	100	60	140	@
43	103	C	C	43	103	63	143	C
45	105	E	E	45	105	65	145	E
46	106	F	F	46	106	66	146	F
49	111	I	I	49	111	69	151	I
4A	112	J	J	4A	112	6A	152	J
4C	114	L	L	4C	114	6C	154	L
4F	117	O	O	4F	117	6F	157	O
51	121	Q	Q	51	121	71	161	Q
52	122	R	R	52	122	72	162	R
54	124	T	T	54	124	74	164	T
57	127	W	W	57	127	77	167	W
58	130	X	X	58	130	78	170	X
5B	133	[	[	5B	133	7B	173	[
5D	135	]	]	5D	135	7D	175	]
5E	136	^	¬	5E	136	7E	176	^
A1	241	!		21	041			!
A2	242	"	≠	22	042			"
A4	244	$	$	24	044			$
A7	247	'		27	047			'
A8	250	(	(	28	050			(

TABLE A-12. CHARACTER CODE TRANSLATIONS, ASCII CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 10 AND 15 (200UT AND 734) (Contd)

Terminal ASCII†				Network ASCII (Normalized Mode Use)				
Hex Code††	Octal Code††	Keyboard or Printer Graphic		Input or Output		Console Output Only		Graphic
		ASCII	CDC	Hex Code†††	Octal Code†††	Hex Code†††	Octal Code†††	
AB	253	+	+	2B	053			+
AD	255	-	-	2D	055			-
AE	256	.	.	2E	056			.
B0	260	0	0	30	060			0
B3	263	3	3	33	063			3
B5	265	5	5	35	065			5
B6	266	6	6	36	066			6
B9	271	9	9	39	071			9
BA	272	:	:	3A	072			:
BC	274	<	<	3C	074			<
BF	277	?	↓	3F	077			?
C1	301	A	A	41	101	61	141	A
C2	302	B	B	42	102	62	142	B
C4	304	D	D	44	104	64	144	D
C7	307	G	G	47	107	67	147	G
C8	310	H	H	48	110	68	150	H
CB	313	K	K	4B	113	6B	153	K
CD	315	M	M	4D	115	6D	155	M
CE	316	N	N	4E	116	6E	156	N
D0	320	P	P	50	120	70	160	P
D3	323	S	S	53	123	73	163	S
D5	325	U	U	55	125	75	165	U
D6	326	V	V	56	126	76	166	V
D9	331	Y	Y	59	131	79	171	Y
DA	332	Z	Z	5A	132	7A	172	Z
DC	334	\	≥	5C	134	7C	174	\
DF	337	_	↱	5E	135	7F	177	_

†Escape codes are not listed.

††Shown with odd parity, the only possible parity selection for these terminal classes. ASCII control codes 000 through 040_8 (without parity) are removed from input during complete editing; codes 01_8 and 03_8 (SOH and ETX, without parity) are preserved as data in full-ASCII mode, as are escape code sequences.

†††Shown with zero parity (eighth or uppermost bit is always zero). During output, codes 000 through 010_8 are converted to code 040_8 (blank); codes 012_8, 015_8, and 037_8 (LF, CR, and US) are removed. 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.

TABLE A-13. CHARACTER CODE TRANSLATIONS, EXTERNAL BINARY CODED (BCD) CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 10 AND 15 (200UT and 734)

Terminal External BCD†				Network ASCII (Normalized Mode Use)				
		Keyboard or Printer Graphic		Input or Output		Console Output Only		
Hex. Code††	Octal Code††	ASCII	CDC	Hex. Code†††	Octal Code†††	Hex. Code†††	Octal Code†††	Graphic
10	020	:	:	3A	072			:
20	040	-	-	2D	055			-
23	043	L	L	4C	114	6C	154	L
25	045	N	N	4E	116	6E	156	N
26	046	O	O	4F	117	6F	157	O
29	051	R	R	52	122	72	162	R
2A	052	!	∨	21	041			!
2C	054	*	*	2A	052			*
2F	057	>	>	3E	076			>
31	061	A	A	41	101	61	141	A
32	062	B	B	42	102	62	142	B
34	064	D	D	44	104	64	144	D
37	067	G	G	47	107	67	147	G
38	070	H	H	48	110	68	150	H
3B	073	.	.	2E	056			.
3D	075			5C	134	7C	174	\
43	103	3	3	33	063			3
45	105	5	5	35	065			5
46	106	6	6	36	066			6
49	111	9	9	39	071			9
4A	112	0	0	30	060			0
4C	114	=	≠	22	042			"
4F	117	[	[	5B	133	7B	173	[
51	121	/	/	2F	057			/
52	122	S	S	53	123	73	163	S
54	124	U	U	55	125	75	165	U
57	127	X	X	58	130	78	170	X
58	130	Y	Y	59	131	79	171	Y
5B	133	,	,	2C	054			,
5D	135	_	↱	5F	137	7F	177	_
5E	136	#	≡	23	043			#
A1	241	J	J	4A	112	6A	152	J
A2	242	K	K	4B	113	6B	153	K
A4	244	M	M	4D	115	6D	155	M
A7	247	P	P	50	120	70	160	P
A8	250	Q	Q	51	121	71	161	Q
AB	253	$	$	24	044			$

TABLE A-13. CHARACTER CODE TRANSLATIONS, EXTERNAL BINARY CODED (BCD) CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 10 AND 15 (200UT and 734) (Contd)

Terminal External BCD†				Network ASCII (Normalized Mode Use)				
Hex. Code††	Octal Code††	Keyboard or Printer Graphic		Input or Output		Console Output Only		Graphic
		ASCII	CDC	Hex. Code†††	Octal Code†††	Hex. Code†††	Octal Code†††	
AD	255	'	↑	27	047			'
AE	256	?	↓	3F	077			?
B3	263	C	C	43	103	63	143	C
B5	265	E	E	45	105	65	145	E
B6	266	F	F	46	106	66	146	F
B9	271	I	I	49	111	69	151	I
BA	272	<	<	3C	074			<
BC	274	)	)	29	051			)
BF	277	;	;	3B	073			;
C1	301	1	1	31	061			1
C2	302	2	2	32	062			2
C4	304	4	4	34	064			4
C7	307	7	7	37	067			7
C8	310	8	8	38	070			8
CB	313	=	=	3D	075			=
CD	315	@	≤	40	100	60	140	@
CE	316	%	%	25	045			%
D0	320	blank	blank	20	040			space
D3	323	T	T	54	124	74	164	T
D5	325	V	V	56	126	76	166	V
D6	326	W	W	57	127	77	167	W
D9	331	Z	Z	5A	132	7A	172	Z
DA	332	]	]	5D	135	7D	175	]
DC	334	(	(	28	050			(
DF	337	&	^	26	046			&
D0	320	^ or blank	¬ or ■ or none			5E, 7E	136, 176	^§

†Escape codes and control codes are not listed.

††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.

§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-14. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 11, 12, AND 13 (711, 714, AND 714X)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character§
73	163	s		73	163	s	
75	165	u		75	165	u	
76	166	v		76	166	v	
79	171	y		79	171	y	
7A	172	z		7A	172	z	
7C	174	¦ or ↑ or |		7C	174	¦	
7F	177	■	DEL or RUBOUT	7F	177		delete
80	200		NUL or @	20	040	space	
83	203		ETX or C	03	003		end of text§§
85	205		ENQ or WRU or E	20	040	space	
86	206		ACK or RU or F	20	040	space	
89	211		HT or I	09	011		horizontal tabulate
8A	212		LF or NL or ↓ or J or NEW LINE	0A	012		linefeed
8C	214		FF or FORM or L	0C	014		formfeed
8F	217		SI or O	0F	017		shift in
91	221		DC1 or X-ON or Q	11	021		device control 1
92	222		DC2 or TAPE or R	12	022		device control 2
94	224		DC4 or TAPE or T	14	024		device control 4
97	227		ETB or W	17	027		end transmission block
98	230		CAN or CLEAR or X	18	030		cancel
9B	233		ESC or ESCAPE or [	1B	033		escape
9D	235		GS or]	1D	035		group separator
9E	236		RS or ^	1E	036		record separator
A1	241	!		21	041	!	
A2	242	"		22	042	"	
A4	244	$		24	044	$	
A7	247	'		27	047	'	
A8	250	(		28	050	(	
AB	253	+		2B	053	+	
AD	255	-		2D	055	-	
AE	256	.		2E	056	.	
B0	260	0		30	060	0	
B3	263	3		33	063	3	
B5	265	5		35	065	5	
B6	266	6		36	066	6	
B9	271	9		39	071	9	
BA	272	:		3A	072	:	
BC	274	<		3C	074	<	
BF	277	?		3F	077	?	
C1	301	A		41	101	A	
C2	302	B		42	102	B	
C4	304	D		44	104	D	
C7	307	G		47	107	G	
C8	310	H		48	110	H	
CB	313	K		4B	113	K	
CD	315	M		4D	115	M	
CE	316	N		4E	116	N	
D0	320	P		50	120	P	
D3	323	S		53	123	S	
D5	325	U		55	125	U	
D6	326	V		56	126	V	
D9	331	Y		59	131	Y	
DA	332	Z		5A	132	Z	
DC	334	\		5C	134	∨	
DF	337	_ or ←		5F	137		
E0	340	`		60	140	`	
E3	343	c		63	143	c	

TABLE A-14. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 11, 12, AND 13 (711, 714, AND 714X) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character§
01	001		SOH or A	01	001		start of header§§
02	002		STX or B	20	040	space	
04	004		EOT or D	20	040	space	
07	007		BELL or G	20	040	space	
08	010		BS or ← or H	20	040	space	
0B	013		VT or K	0B	013		vertical tabulate
0D	015		CR or RETURN or M				
0E	016		SO or N	0E	016		shift out
10	020		DLE or P	10	020		data link escape
13	023		DC3 or X-OFF or S	13	023		device control 3
15	025		NAK or → or U	15	025		negative acknowledgment
16	026		SYN or LINE CLEAR or V	16	026		synchronous idle
19	031		EM or RESET or Y	19	031		end of medium
1A	032		SUB or ↑ or Z	1A	032		substitute
1C	034		FS or \	1C	034		file separator
1F	037		US or -	20	040	space	
20	040	SPACE or blank		20	040	space	
23	043	#		23	043	#	
25	045	%		25	045	%	
26	046	&		26	046	&	
29	051	)		29	051	)	
2A	052	*		2A	052	*	
2C	054	,		2C	054	,	
2F	057	/		2F	057	/	
31	061	1		31	061	1	
32	062	2		32	062	2	
34	064	4		34	064	4	
37	067	7		37	067	7	
38	070	8		38	070	8	
3B	073	;		3B	073	;	
3D	075	=		3D	075	=	
3E	076	>		3E	076	>	
40	100	@		40	100	@	
43	103	C		43	103	C	
45	105	E		45	105	E	
46	106	F		46	106	F	
49	111	I		49	111	I	
4A	112	J		4A	112	J	
4C	114	L		4C	114	L	
4F	117	O		4F	117	O	
51	121	Q		51	121	Q	
52	122	R		52	122	R	
54	124	T		54	124	T	
57	127	W		57	127	W	
58	130	X		58	130	X	
5B	133	[		5B	133	[	
5D	135	]		5D	135	]	
5E	136	^ or ¬		5E	136	^	
61	141	a		61	141	a	
62	142	b		62	142	b	
64	144	d		64	144	d	
67	147	g		67	147	g	
68	150	h		68	150	h	
6B	153	k		6B	153	k	
6D	155	m		6D	155	m	
6E	156	n		6E	156	n	
70	160	p		70	160	p	

TABLE A-14. CHARACTER CODE TRANSLATIONS, CONSOLES AND LINE PRINTERS IN TERMINAL CLASSES 11, 12, AND 13 (711, 714, AND 714X) (Contd)

Terminal ASCII (Transparent Mode Use)				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	ASCII Graphic	Control Character††	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character§
E5	345	e		65	145	e	
E6	346	f		66	146	f	
E9	351	i		69	151	i	
EA	352	j		6A	152	j	
EC	354	l		6C	154	l	
EF	357	o		6F	157	o	
F1	361	q		71	161	q	
F2	362	r		72	162	r	
F4	364	t		74	164	t	
F7	367	w		77	167	w	
F8	370	x		78	170	x	
FB	373	{		7B	173	{	
FD	375	}		7D	175	}	
FE	376	~ or ¬		7E	176	~	

†Shown with odd parity, the only possible parity selection for these terminal classes.

††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).

§Converted to a space (040_8) within a batch printer file.

§§Converted to a space (040_8) during complete editing.

TABLE A-15. ASCII CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal EBCD				Network ASCII (Normalized Mode Use)			
Hex. Code[†]	Octal Code[†]	EBCD Graphic[††]	Control Character	Hex. Code[†††]	Octal Code[†††]	ASCII Graphic	Control Character
01	001	_ or -		5F or 2D	137 or 055	_ or -	
02	002	¢ or @		21 or 40	140 or 100	` or @	
04	004	* or 8		2A or 38	052 or 070	* or 8	
07	007	H or h		48 or 68	110 or 150	H or h	
08	010	: or 4		3A or 34	072 or 064	: or 4	
0B	013	D or d		44 or 64	104 or 144	D or d	
0D	015		RES or RESTORE	00	000		null
0E	016		BY or BYPASS	00	000		null
10	020	< or 2		3C or 32	074 or 062	< or 2	
13	023	B or b		42 or 62	102 or 142	B or b	
15	025		undefined	00	000		null
16	026		undefined	00	000		null
19	031	O or o		4F or 6F	117 or 157	O or o	
1A	032	W or w		57 or 77	127 or 167	W or w	
1C	034		UCS or UPPERCASE	0E	016		shift out[§]
1F	037		LCS or LOWERCASE	0F	017		shift in[§]
20	040	= or 1		3D or 31	075 or 061	= or 1	
23	043	A or a		41 or 61	101 or 141	A or a	
25	045	R or r		52 or 72	122 or 162	R or r	
26	046	Z or z		5A or 7A	132 or 172	Z or z	
29	051	N or n		4E or 6E	116 or 156	N or n	
2A	052	V or v		56 or 76	126 or 166	V or v	
2C	054		RO or READER STOP	14	024		device control 4
2F	057		HT or TAB	09	011		horizontal tabulate
31	061	L or l		4C or 6C	114 or 154	L or l	
32	062	T or t		54 or 74	124 or 164	T or t	
34	064	" or #		22 or 23	042 or 043	= or #	
37	067	¬ or .		5E or 2E	136 or 056	^ or .	
38	070	> or 7		3E or 37	076 or 067	> or 7	
3B	073	G or g		47 or 67	107 or 147	G or g	
3D	075		IL or IDLE or NULL	00	000		null
3E	076		PRE or PREFIX	01	001		start of header[§]
40	100	space		20	040	space	
43	103	+ or &		2B or 26	053 or 046	+ or &	
45	105	Q or q		51 or 71	121 or 161	Q or q	
46	106	Y or y		59 or 79	131 or 171	Y or y	
49	111	M or m		4D or 6D	115 or 155	M or m	
4A	112	U or u		55 or 75	125 or 165	U or u	
4C	114		PN or PUNCH ON	11	021		device control 1 (tape on)
4F	117		PF or PUNCH OFF	13	023		device control 3 (tape off)
51	121	K or k		4B or 6B	113 or 153	K or k	
52	122	S or s		53 or 73	123 or 163	S or s	
54	124	) or 0		29 or 30	051 or 060	) or 0	
57	127		undefined	00	000		null
58	130	' or 6		27 or 36	047 or 066	' or 6	
5B	133	F or f		46 or 66	106 or 146	F or f	
5D	135		BS or BACKSPACE	08	010		backspace
5E	136		EOB	17	027		end transmission block[§]
61	141	J or j		4A or 6A	112 or 152	J or j	
62	142	? or /		3F or 2F	077 or 057	? or /	
64	144	(or 9		28 or 39	050 or 071	(or 9	
67	147	I or i		49 or 69	111 or 151	I or i	
68	150	% or 5		25 or 35	045 or 065	% or 5	
6B	153	E or e		45 or 65	105 or 145	E or e	
6D	155		NL or CR or RETURN	0D	015		carriage return
6E	156		LF or LINE FEED	0A	012		linefeed

TABLE A-15. ASCII CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741) (Contd)

Terminal EBCD				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	EBCD Graphic††	Control Character	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
70	160	; or 3		3B or 33	073 or 063	; or 3	
73	163	C or c		43 or 63	103 or 143	C or c	
75	165	! or $		21 or 24	041 or 044	! or $	
76	166	| or ,		7C or 2C	174 or 054	¦ or ,	
79	171	P or p		50 or 70	120 or 160	P or p	
7A	172	X or x		58 or 78	130 or 170	X or x	
7C	174		EOT	04	004		end of transmission§
7F	177		DEL	7F	177		delete
00	000	space		5B thru 5D	133 thru 135	[or \ or]	
00	000	space		60	140	`	
00	000	space		7B	173	{	
00	000	space		7D or 7E	175 or 176	} or ~	
3D	075		IL or IDLE or NULL§§	02	002		start of text
3D	075		IL or IDLE or NULL§§	03	003		end of text
3D	075		IL or IDLE or NULL§§	05	005		enquire
3D	075		IL or IDLE or NULL§§	07	007		bell
3D	075		IL or IDLE or NULL§§	0B or 0C	013 or 014		vertical tabulate or formfeed
3D	075		IL or IDLE or NULL§§	10	020		data link escape
3D	075		IL or IDLE or NULL§§	12	022		device control 2
3D	075		IL or IDLE or NULL§§	14 thru 16	024 thru 026		device control 4, negative acknowledge, or synchronize
3D	075		IL or IDLE or NULL§§	18 thru 1F	030 thru 037		cancel, end of media, substitute, escape, file separator, group separator, record separator, or unit separator

†Shown with odd parity; odd parity is the default for this terminal class.

††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-16. APL CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal EBCD-APL				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	EBCD-APL Graphic††	Control Character	Hex. Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
01	001	_ or +		5F or 2D	137 or 053	_ or +	
02	002	→ or ←		71 or 70	161 or 160	→ or ←	
04	004	≠ or 8		22 or 38	042 or 070	≠ or 8	
07	007	Δ or H		68 or 48	150 or 110	Δ or H	
08	010	≤ or 4		40 or 34	100 or 064	≤ or 4	
0B	013	⌊ or D		64 or 44	144 or 104	⌊ or D	
0D	015		undefined	00	000		null
0E	016		undefined	00	000		null
10	020	- or 2		2D or 32	055 or 062	- or 2	
13	023	⊥ or B		42 or 62	142 or 102	⊥ or B	
15	025		undefined	00	000		null
16	026		undefined	00	000		null
19	031	○ or O		6F or 4F	157 or 117	○ or O	
1A	032	ω or W		77 or 57	167 or 127	ω or W	
1C	034		UCS or UPPERCASE	0E	016		shift out§
1F	037		LCS or LOWERCASE	0F	017		shift in§
20	040	" or 1		22 or 31	042 or 061	" or 1	
23	043	α or A		61 or 41	141 or 101	α or A	
25	045	ρ or R		72 or 52	162 or 122	ρ or R	
26	046	⊂ or Z		7A or 5A	172 or 132	⊂ or Z	
29	051	⊤ or N		6E or 4E	156 or 116	⊤ or N	
2A	052	∪ or V		76 or 56	166 or 126	∪ or V	
2C	054		undefined	00	000		null
2F	057		HT or TAB	06	006		horizontal tabulate
31	061	⎕ or L		6C or 4C	154 or 114	⎕ or L	
32	062	~ or T		74 or 54	164 or 124	~ or T	
34	064	) or]		29 or 5D	051 or 135	) or]	
37	067	: or .		3A or 2E	072 or 056	: or .	
38	070	> or 7		3E or 37	076 or 067	> or 7	
3B	073	∇ or G		67 or 47	147 or 107	∇ or G	
3D	075		IL or IDLE or NULL	00	000		null
3E	076		PRE or PREFIX	1B	033		escape
40	100	space		20	040	space	
43	103	+ or X		25 or 66	045 or 146	+ or X	
45	105	? or Q		3F or 51	077 or 121	? or Q	
46	106	↑ or Y		79 or 59	171 or 131	↑ or Y	
49	111	| or M		6D or 4D	155 or 115	| or M	
4A	112	↓ or U		75 or 55	165 or 125	↓ or U	
4C	114		undefined	00	000		null
4F	117		undefined	00	000		null
51	121	⊣ or K		6B or 4B	153 or 113	⊣ or K	
52	122	⌈ or S		73 or 53	163 or 123	⌈ or S	
54	124	∧ or 0		26 or 30	046 or 060	∧ or 0	
57	127		undefined	00	000		null
58	130	≥ or 6		7C or 36	174 or 066	≥ or 6	
5B	133	= or F		5E or 46	136 or 106	= or F	
5D	135		BS or BACKSPACE	08	010		backspace
5E	136		EOB	17	027		end transmission block§
61	141	∘ or J		6A or 4A	152 or 112	∘ or J	
62	142	\ or /		5C or 2F	134 or 057	\ or /	
64	144	∨ or 9		21 or 39	041 or 071	∨ or 9	
67	147	⍳ or I		69 or 49	151 or 111	⍳ or I	
68	150	= or 5		3D or 35	075 or 065	= or 5	
6B	153	∊ or E		65 or 45	145 or 105	∊ or E	
6D	155		NL or CR or RETURN	0D	015		carriage return
6E	156		LF or LINE FEED	0A	012		line feed
70	160	< or 3		3C or 33	074 or 063	< or 3	
73	163	∩ or C		63 or 43	143 or 103	∩ or C	
75	165	(or [		28 or 5B	050 or 133	(or [	
76	166	; or ,		3B or 2C	073 or 054	; or ,	
79	171	* or P		2A or 50	052 or 120	* or P	
7A	172	⊃ or X		78 or 58	170 or 130	⊃ or X	

TABLE A-16. APL CHARACTER CODE TRANSLATIONS, EBCD CONSOLES IN TERMINAL CLASS 4 (2741) (Contd)

Terminal EBCD-APL				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	EBCD-APL Graphic††	Control Character	Hex. Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
7C	174		EOT	04	004		end of transmission§
7F	177		DEL	7F	177		delete
00	000	space§§		27	047	'	
00	000	space§§		60	140	◇	
00	000	space§§		7B	173	{	
00	000	space§§		7D	175	}	
3D	075		IL or IDLE or NULL§§	02	002		start of text
3D	075		IL or IDLE or NULL§§	03	003		end of text
3D	075		IL or IDLE or NULL§§	05	005		enquire
3D	075		IL or IDLE or NULL§§	07	007		bell
3D	075		IL or IDLE or NULL§§	0B or 0C	013 or 014		vertical tabulate or form feed
3D	075		IL or IDLE or NULL§§	10 thru 16	020 thru 026		data link escape, device control 1 thru device control 4, negative acknowledge, or synchronize
3D	075		IL or IDLE or NULL§§	18 thru 1F	030 thru 037		cancel, end of media, substitute, escape file separator, group separator, record separator, or unit separator

†Shown with odd parity; odd parity is the default for this terminal class.

††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. ASCII CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal Correspondence Code				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	Correspondence Code Graphic††	Control Character	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
01	001	1/4 or 1/2		5B or 5D	137 or 135	[or]	
02	002	T or t		54 or 74	124 or 164	T or t	
04	004	$ or 4		24 or 34	044 or 064	$ or 4	
07	007	? or /		3F or 2F	077 or 057	? or /	
08	010	% or 5		25 or 35	045 or 065	% or 5	
0B	013	P or p		50 or 70	120 or 160	P or p	
0D	015		RES or RESTORE	00	000		null
0E	016		BY or BYPASS	00	000		null
10	020	@ or 2		40 or 32	100 or 062	@ or 2	
13	023	+ or =		2B or 3D	053 or 075	+ or =	
15	025		undefined	00	000		null
16	026		undefined	00	000		null
19	031	I or i		49 or 69	111 or 151	I or i	
1A	032	K or k		4B or 6B	113 or 153	K or k	
1C	034		UCS or UPPERCASE	0E	016		shift out§
1F	037		LCS or LOWERCASE	0F	017		shift in§
20	040	± or 1		7C or 31	174 or 061	¦ or 1	
23	043	G or g		47 or 67	107 or 147	G or g	
25	045	S or s		53 or 73	123 or 163	S or s	
26	046	H or h		48 or 68	110 or 150	H or h	
29	051	R or r		52 or 72	122 or 162	R or r	
2A	052	D or d		44 or 64	104 or 144	D or d	
2C	054		RO or READER STOP	14	024		device control 4
2F	057		HT or TAB	09	011		horizontal tabulate
31	061	V or v		56 or 76	126 or 166	V or v	
32	062	U or u		55 or 75	125 or 165	U or u	
34	064	(or 9		28 or 39	050 or 071	(or 9	
37	067	_ or -		5F or 2D	137 or 055	_ or -	
38	070	* or 8		2A or 38	052 or 070	* or 8	
3B	073	,		2C	054	,	
3D	075		IL or IDLE or NULL	00	000		null
3E	076		PRE or PREFIX	1B	033		escape
40	100	space		20	040	space	
43	103	J or j		4A or 6A	112 or 152	J or j	
45	105	O or o		4F or 6F	117 or 157	O or o	
46	106	L or l		4C or 6C	114 or 154	L or l	
49	111	" or '		22 or 27	042 or 041	" or '	
4A	112	E or e		45 or 65	105 or 145	E or e	
4C	114		PN or PUNCH ON	11	021		device control 1 (tape on)
4F	117		PF or PUNCH OFF	13	023		device control 3 (tape off)
51	121	.		2E	056	.	
52	122	N or n		4E or 6E	116 or 156	N or n	
54	124	Z or z		5A or 7A	132 or 172	Z or z	
57	127		undefined	00	000		null
58	130	¢ or 6		21 or 36	041 or 066	! or 6	
5B	133	Q or q		51 or 71	121 or 161	Q or q	
5D	135		BS or BACKSPACE	08	010		backspace
5E	136		EOB	17	027		end transmission block§
61	141	M or m		4D or 6D	115 or 155	M or m	
62	142	X or x		58 or 78	130 or 170	X or x	
64	144	) or 0		29 or 30	051 or 060	) or 0	
67	147	Y or y		79 or 59	131 or 171	Y or y	
68	150	& or 7		26 or 37	046 or 067	& or 7	
6B	153	: or ;		3A or 3B	072 or 073	: or ;	
6D	155		NL or CR or RETURN	0D	015		carriage return
6E	156		LF or LINE FEED	0A	012		line feed
70	160	# or 3		23 or 33	043 or 063	# or 3	
73	163	F of f		46 or 66	106 or 146	F or f	
75	165	W or w		57 or 77	127 or 167	W or w	

TABLE A-17. ASCII CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741) (Contd)

Terminal Correspondence Code				Network ASCII (Normalized Mode Use)			
Hex. Code†	Octal Code†	Correspondence Code Graphic††	Control Character	Hex. Code†††	Octal Code†††	ASCII Graphic	Control Character
76	166	B or b		42 or 62	102 or 142	B or b	
79	171	A or a		41 or 61	101 or 141	A or a	
7A	172	C or c		43 or 63	103 or 143	C or c	
7C	174		EOT	04	004		end of transmission§
7F	177		DEL	18	030		cancel
00	000	space§§		27	047	'	
00	000	space§§		5C	134	\	
00	000	space§§		5E	136	^	
00	000	space§§		60	140	`	
00	000	space§§		7B	173	{	
00	000	space§§		7D or 7E	175 or 176	} or ~	
3D	075		IL or IDLE or NULL§§	01	001		start of header
3D	075		IL or IDLE or NULL§§	02	002		start of text
3D	075		IL or IDLE or NULL§§	03	003		end of text
3D	075		IL or IDLE or NULL§§	05	005		enquire
3D	075		IL or IDLE or NULL§§	07	007		bell
3D	075		IL or IDLE or NULL§§	0B or 0C	013 or 014		vertical tabulate or form feed
3D	075		IL or IDLE or NULL§§	10	020		data link escape
3D	075		IL or IDLE or NULL§§	12	022		device control 2
3D	075		IL or IDLE or NULL§§	14 thru 16	024 thru 026		device control 4, negative acknowledge, or synchronize
3D	075		IL or IDLE or NULL§§	18 thru 1F	030 thru 037		cancel, end of media, substitute, file separator, group separator, record separator, or unit separator

†Shown with odd parity; odd parity is the default for this terminal class.

††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. APL CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741)

Terminal Correspondence Code				Network ASCII (Normalized Mode Use)			
Hex Code†	Octal Code†	Correspondence Code APL Graphic††	Control Character	Hex Code†††	Octal Code†††	ASCII-APL Graphic	Control Character
01	001	→ or ←		71 or 70	161 or 160	→ or ←	
02	002	~ or T		74 or 54	164 or 124	~ or T	
04	004	≤ or 4		40 or 34	100 or 064	≤ or 4	
07	007	⍀ or /		5C or 2F	134 or 057	⍀ or /	
08	010	= or 5		3D or 35	075 or 065	= or 5	
0B	013	* or P		2A or 50	052 or 120	* or P	
0D	015		undefined	00	000		null
0E	016		undefined	00	000		null
10	020	¯ or 2		5E or 32	136 or 062	¯ or 2	
13	023	+ or X		25 or 66	045 or 146	+ or X	
15	025		undefined	00	000		null
16	026		undefined	00	000		null
19	031	⍳ or I		69 or 49	151 or 111	⍳ or I	
1A	032	' or K		27 or 4B	153 or 113	' or K	
1C	034		UCS or UPPERCASE	0E	016		shift out§
1F	037		LCS or LOWERCASE	0F	017		shift in§
20	040	" or 1		23 or 31	042 or 061	" or 1	
23	043	∇ or G		67 or 47	147 or 107	∇ or G	
25	045	⌈ or S		73 or 53	163 or 123	⌈ or S	
26	046	Δ or H		68 or 48	150 or 110	Δ or H	
29	051	⍴ or R		72 or 52	162 or 122	⍴ or R	
2A	052	⌊ or D		64 or 44	144 or 104	⌊ or D	
2C	054		undefined	00	000		null
2F	057		HT or TAB	09	011		horizontal tabulate
31	061	∪ or V		76 or 56	166 or 126	∪ or V	
32	062	↓ or U		75 or 55	165 or 125	↓ or U	
34	064	∨ or 9		21 or 39	041 or 071	∨ or 9	
37	067	- or +		2D or 2B	055 or 053	- or +	
38	070	≠ or 8		22 or 38	042 or 070	≠ or 8	
3B	073	; or ,		3B or 2C	073 or 054	; or ,	
3D	075		IL or IDLE or NULL	00	000		null
3E	076		PRE or PREFIX	1B	033		escape
40	100	space		20	040	space	
43	103	∘ or J		6A or 4A	156 or 112	∘ or J	
45	105	○ or O		6F or 4F	157 or 117	○ or O	
46	106	⎕ or L		6C or 4C	154 or 114	⎕ or L	
49	111	) or]		29 or 5D	051 or 035	) or]	
4A	112	∊ or E		65 or 45	145 or 105	∊ or E	
4C	114		undefined	00	000		null
4F	117		undefined	13	023		null
51	121	: or .		3A or 2E	072 or 056	: or .	
52	122	⊤ or N		6E or 4E	156 or 116	⊤ or N	
54	124	⊂ or Z		7A or 5A	172 or 132	⊂ or Z	
57	127		undefined	00	000		null
58	130	≥ or 6		7C or 36	174 or 066	≥ or 6	
5B	133	? or Q		3F or 51	077 or 121	? or Q	
5D	135		BS or BACKSPACE	08	010		backspace
5E	136		EOB	17	027		end transmission block§
61	141	∣ or M		6D or 4D	155 or 115	∣ or M	
62	142	⊃ or X		78 or 58	170 or 130	⊃ or X	
64	144	∧ or 0		26 or 30	045 or 060	∧ or 0	
67	147	↑ or Y		79 or 59	171 or 131	↑ or Y	
68	150	> or 7		3E or 37	076 or 067	> or 7	
6B	153	(or [		28 or 5B	050 or 133	(or [	
6D	155		NL or CR or RETURN	0D	015		carriage return
6E	156		LF or LINE FEED	0A	012		line feed
70	160	< or 3		3C or 33	074 or 063	< or 3	
73	163	_ or F		5F or 46	137 or 106	_ or F	
75	165	⍵ or W		77 or 57	167 or 127	⍵ or W	

TABLE A-18. APL CHARACTER CODE TRANSLATIONS, CORRESPONDENCE CODE CONSOLES IN TERMINAL CLASS 4 (2741) (Contd)

Terminal Correspondence Code				Network ASCII (Normalized Mode Use)			
Hex Code[†]	Octal Code[†]	Correspondence Code APL Graphic[††]	Control Character	Hex Code[†††]	Octal Code[†††]	ASCII-APL Graphic	Control Character
76	166	⊥ or B		62 or 42	142 or 102	⊥ or B	
79	171	α or A		61 or 41	141 or 101	α or A	
7A	172	∩ or C		63 or 43	143 or 103	∩ or C	
7C	174		EOT	04 or 14	004		end of transmission[§]
7F	177		DEL	18	030		cancel
00	000	space[§§]		27	047	'	
00	000	space[§§]		60	140	◇	
00	000	space[§§]		7B	173	{	
00	000	space[§§]		7D or 7E	175 or 176	} or ⊢	
3D	075		IL or IDLE or NULL[§§]	01	001		start of header
3D	075		IL or IDLE or NULL[§§]	02	002		start of text
3D	075		IL or IDLE or NULL[§§]	03	003		end of text
3D	075		IL or IDLE or NULL[§§]	05	005		enquire
3D	075		IL or IDLE or NULL[§§]	07	007		bell
3D	075		IL or IDLE or NULL[§§]	0B or 0C	013 or 014		vertical tabulate or form feed
3D	075		IL or IDLE or NULL[§§]	10	020		data link escape
3D	075		IL or IDLE or NULL[§§]	12	022		device control 2
3D	075		IL or IDLE or NULL[§§]	14 thru 16	024 thru 026		device control 4, negative acknowledge, or synchronize
3D	075		IL or IDLE or NULL[§§]	18 thru 1F	030 thru 037		cancel, end of media, substitute, file separator, group separator, record separator, or unit separator

[†]Shown with odd parity; odd parity is the default for this terminal class. (Unless PA=N or PA=I, 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. FULL ASCII NORMALIZED MODE APL CHARACTER SET

								128-Character Set							
										96-Character Subset					
										64-Character Subset					
Bits						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
b_4	b_3	b_2	b_1	ROW	COLUMN			0	1	2	3	4	5	6	7
0	0	0	0	0				NUL 000	DLE 020	SP 040	0 060	≤ 100	P 120	◇ 140	← 160
0	0	0	1	1				SOH 001	DC1 021	¨ or ∨ 041	1 061	A 101	Q 121	∝ 141	→ 161
0	0	1	0	2				STX 002	DC2 022	≠ 042	2 062	B 102	R 122	⊥ 142	⍴ 162
0	0	1	1	3				ETX 003	DC3 023	: 043	3 063	C 103	S 123	∩ 143	⌈ 163
0	1	0	0	4				EOT 004	DC4 024	$ 044	4 064	D 104	T 124	⌊ 144	~ 164
0	1	0	1	5				ENQ 005	NAK 025	÷ 045	5 065	E 105	U 125	∊ 145	↓ 165
0	1	1	0	6				ACK 006	SYN 026	∧ 046	6 066	F 106	V 126	χ 146	∪ 166
0	1	1	1	7				BEL 007	ETB 027	' 047	7 067	G 107	W 127	∇ 147	⍵ 167
1	0	0	0	8				BS 010	CAN 030	(050	8 070	H 110	X 130	Δ 150	⊃ 170
1	0	0	1	9				HT 011	EM 031	) 051	9 071	I 111	Y 131	⍳ 151	↑ 171
1	0	1	0	A				LF 012	SUB 032	* 052	: 072	J 112	Z 132	∘ 152	⊂ 172
1	0	1	1	B				VT 013	ESC 033	+ 053	; 073	K 113	[133	⊣ 153	{ 173
1	1	0	0	C				FF 104	FS 034	, 054	< 074	L 114	\ 134	⎕ 154	≥ 174
1	1	0	1	D				CR 015	GS 035	- 055	= 075	M 115	] 135	| 155	} 175
1	1	1	0	E				SO 016	RS 036	. 056	> 076	N 116	¯ 136	⊤ 156	⊢ 176
1	1	1	1	F				SI 017	US 037	/ 057	? 077	O 117	_ 137	○ 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.

DIAGNOSTIC MESSAGES B

This appendix lists the following categories of messages concerning network software:

Application program execution errors

Application program macro assembly errors

Postprocessor errors and informative messages

EXECUTION ERROR MESSAGES

When the Network Access Method's execution time code detects an 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 ERROR MESSAGES

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 macros are described in section 4.

POSTPROCESSOR MESSAGES

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. The debug log file postprocessor is described in section 6.

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
APPLICATION IS NOT ALLOWED TO DO XFR	The application attempted a call to the AIP routine NETXFR but is not validated for such a call.	Remove the call to NETXFR. Only PTF and QTF are allowed to call NETXFR.	AIP

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
BKSP ERROR ON FILE xxxxxxx - AT=yyB.	AIP encountered an I/O error while backspacing the specified file one record; yy is the abnormal termination code returned by CIO (nonfatal).	Check the abnormal termination code to determine what is wrong with the file and then correct the problem.	AIP
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 system validation file and/or COMTNAP.	Correct the aname parameter and rerun the job. Check that the system validation file and/or COMTNAP 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
OVER 500 SUP MSGS QUEUED FOR APP	The application program is not fetching the asynchronous supervisory messages queued for it. When the queue in NIP reaches 500 supervisory messages, NIP aborts the application program and this dayfile message appears in the application's dayfile.	Correct the program and rerun the job.	NIP

Message	Significance	Action	Issued By
READ ERROR ON FILE xxxxxxx - AT=yyB.	AIP encountered an I/O error while reading the specified file; yy is the abnormal termination code returned by CIO (nonfatal).	Check the abnormal termination code to determine what is wrong with the file and then correct the problem.	AIP
REWIND ERROR ON FILE xxxxxxx - AT=yyB.	AIP encountered an I/O error while rewinding the specified file; yy is the abnormal termination code returned by CIO (nonfatal).	Check the abnormal termination code to determine what is wrong with the file and then correct the problem.	AIP
ROUTE ERROR ON FILE ZZZZZDN - AT=yyB.	AIP encountered an error when it tried to route the ZZZZZDN file to the input queue; yy is the abnormal termination code returned by DSP (nonfatal).	Check the error code to determine why the route failed and then correct the problem.	AIP
SECURITY VIOLATION	The application program has attempted to call NETON as a supervisory or validation program (CS, NS, or NVF).	Change the program's origin type permission and rerun the job.	NIP
WRITE ERROR ON FILE xxxxxxx - AT=yyB.	AIP encountered an I/O error while writing to the specified file; yy is the abnormal termination code returned by CIO (nonfatal).	None. The file is returned to the system and a new one is created.	AIP

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 the program 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		Reserved by CDC.		
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
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

TABLE B-2. APPLICATION PROGRAM DAYFILE AIP AND QTRM DIAGNOSTIC MESSAGES (Contd)

Reason Code	Message	Significance	Action	Issued By
45	NETOFF: NETOFF DURING FILE TRANSFER	Application NETOFF while there is a file transfer still in progress.	Relocate the improperly placed OFF-type call and rerun the job.	AIP
46 thru 48		Reserved by CDC.		
49	NP$LOC: NO ENTRY WITH MATCHING ACN	No entry in file transferring table matching this ACN.	Rerun the job.	AIP
50	NP$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-CONNS FIELD IS ZERO	The num-conns 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 host operator has disabled the application from accessing the network.	Rerun the job after contacting the host 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 A-TO-A	The application program requested connection to another application program when the A-to-A field is not set.	Change the program to set the A-to-A field before the call to QTOPEN and rerun the job.	QTRM
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 NETGTFL 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

TABLE B-3. AIP MACRO ASSEMBLY LISTING DIAGNOSTIC MESSAGES (Contd)

Message	Significance	Action	Issued By
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
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.	Follow site-defined procedure to report and correct product or system problems.	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 DLFP command.	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

TABLE B-4. DLFP DAYFILE, ERROR, AND INFORMATIVE MESSAGES (Contd)

Message	Significance	Action	Issued By
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
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

TABLE B-4. DLFP DAYFILE, ERROR, AND INFORMATIVE MESSAGES (Contd)

Message	Significance	Action	Issued By
ILLEGAL PARAMETER	DLFP does not recognize a parameter in the command.	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 between them.	Correct and rerun.	DLFP
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 separators 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 DLFP command is not formatted correctly.	Correct and rerun.	DLFP
PARAMETER SPECIFIED TWICE	A parameter on the DLFP command 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

GLOSSARY C

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 and special graphic 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 network data 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, teletypewriter-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.

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. Card readers, line printers, and plotters are examples of batch devices. Batch devices are sometimes referred to as passive devices.

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 configured 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 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 Type -
See Application Block Type.

Break -
A method employed by a terminal operator to interrupt output or input in progress.

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 irrelevant codes.

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.

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 use 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.

Cluster -
Mode 4 devices grouped by a common cluster address. Synonymous with terminal.

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
Terminal	Equipment controller	Station
Device	Equipment	Device

Communication Element -
Any entity that constitutes a point of input to, or output from, the data communication network. This includes terminal devices, 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.

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. 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 Network Configuration.

Connection -
See Logical Connection.

Connection Number (CN) -
A unique number assigned to each active device on a logical link.

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), multiplexer 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, macroassembler, and microassembler languages. The compiled or assembled output of the Cross programs are in object code format on host computer 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.

Data -
Any portion of a message created by the source, exclusive of any information used to accomplish transmission of such a message.

Debugging -
The process of altering a program to rid it of anomalies.

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 command 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 block. For instance, the DN of an upline block may be the host process which passes the block to the application program responsible for processing the block.

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.

Downline -
The direction of output information flow, from a host computer application program.

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.

Echo -
The process of displaying a keystroke on a 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 NPU's host side and transmission blocks on the NPU's terminal side. A file contains one or more records. Example: a card reader job consists of a file containing the card image records of all the cards in the job deck.

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 line. 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 network data 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 multiplex subsystem which uses a hardware-controlled frame on the input and output multiplex 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 that 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.

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 is stopped by its software. 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 PRU blocks.

Header -
The portion or portions of a block holding information about the block source, destination, and type. During network movement, a block can acquire several headers. For example, during movement of a block from a terminal to the host over an X.25 network, the block acquires the following headers: one at the terminal (also a trailer), one for the frame, one for the packet, and another for the host application program. Headers are discarded by the appropriate stage of processing, so that in this example, the host sees only the application program 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 application programs that process network blocks.

Host Interface Package (HIP) -
The CCP program that handles block transfers across the host/local NPU interface. The HIP transfers control blocks and data blocks (IVT blocks or PRU blocks).

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 functions as well as those functions unique to the HOP despite the existance of NPU operators. There can be only one HOP. Contrast with NPU operator.

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 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 device. An interactive 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 consoles. CCP TIPs convert all upline interactive blocks to this format (exception: no transformations are made to transparent data except to put the data 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.

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.

Link -
A connection between two NPUs or an NPU and a host.

Link Interface Package (LIP) -
The CCP program that 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 multiplexer 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 logical relationships among the service elements in the network. The file contains a list of the application programs available for execution in the host computer, and the users that require automatic login to them. This is a NOS direct access permanent file.

Local NPU -
An NPU that 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 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 console device. 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 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 that 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 an X.25 PAD service.

Macromemory -
The portion of 255x Series network processing unit memory that contains code involved in data communication, such as the Terminal Interface Program. 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.

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 8192 words of 64-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 that 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. Telephone lines interface to digital equipment via modems. Modem is synonymous with data set.

Module -
See Program.

Monitor -
The portion of the CCP 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 that 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 that 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.

Network -
An interconnected set of network processing units, hosts, and terminal devices.

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 -
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.

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 Communications 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 -
The network operator who resides at a terminal and controls network elements such as NPUs, trunks, logical links, lines, and terminals. Contrast with Host Operator. Also, an operator using the offnet NPU console.

Off-Line Diagnostics -
Optional diagnostics for the NPU that require the NPU to 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 the host.

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 Service (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 PSN. Disassembly: The encoding of blocks for transmission to an asynchronous terminal.

Packet-Switching Network (PSN) -
A network that 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) service. 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 packet-switching network.

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.

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 the Pascal language.

Passive Device -
Any device 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 devices. Also known as a nonconsole device. Contrast with Interactive Device.

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 freewheeling asynchronous terminals.

Port -
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 the current 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 devices out of all devices producing network traffic. Terminals with priority are the last devices for which network traffic is suspended when traffic must be temporarily stopped because the network is operating at capacity. Devices with priority receive preferential treatment of their input or output.

Program Initiation Control Block (PICB) -
A program initiation control block consisting of a sequence of commands that control NPU load or dump operations for a specific NPU variant. Several PICB's may exist on the network load file, each as a separate record with a unique NPU variant name as its record name.

Protocol -
A set of standardized conventions that 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 Block (PRUB) -
Physical record unit block. A block format for batch devices that is compatible with the host's PRU (batch file) handling capabilities. CCP TIPs convert all upline batch data 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 that 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.

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.

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; individual records are accessed by their relative PRU numbers.

Record -
(1) A data unit defined for the host record manager; (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 multiplex 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 device-by-device 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, the connection number is 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 network data block.

String -
A unit of information transmission. One or more strings compose a record. A string can be composed of different characters or contiguous identical characters.

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.

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 character synchronization is achieved by recognition of a predefined sync character that precedes the block of data.

Terminal -
An entity, external to the data communication network but connected to it via a communication line, that supplies input to, and/or accepts output from, an application program. In the context of this manual, a terminal is each separately addressable group of devices comprising a physical terminal or station.

Terminal Address -
The hardware address of a mode 4 console, a mode 4C printer or a 3780 card punch. 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 Definition 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.

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 7-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 device to the network 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 host central memory words of the supervisory message or network data block that the application program will accept for processing.

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 data block'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 7-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 that uses the frame as the basic communications element.

Typeahead (Terminal) -
The ability of a terminal to enter input data at all times. The ASYNC TIP supports typeahead; the X.25 TIP supports typeahead if it is provided by the PSN.

Upline -
The direction of input flow to a host computer application program.

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 PSN.

Word -
The basic storage and processing element of a computer. The NPU uses 16-bit words (main memory) and 32-bit word (internal to the micro processor 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.

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 packet-switching network. It is characterized by high-speed, framed data transfers over links. A PSN requires a PAD access for attaching asynchronous terminals.

X.25 TIP -
The CCP TIP that interfaces an NPU to a packet-switching 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
ACN	Application Connection Number
ACT	Application Character Type
AIP	Application Interface Program
ALN	Application List Number
ANAME	Application Name
APL	A Programming Language
ASCII	American Standard Code for Information Interchange
ASYNC	Asynchronous
BCD	Binary Coded Decimal
BIP	Block Interface Package
BLK	Message Block
BRK	Break Block
BSC	Binary Synchronous Communication
BT	Block Type
B1, B2	User-defined breaks
CA	Cluster Address
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
CIB	Circular Input Buffer
CLA	Communications Line Adapter
CMD	Command Block
CR	Carriage Return
CRC	Cyclic Redundancy Check
CRT	Cathode Ray Tube
CS	Communications Supervisor

DBC	Data Block Clarifier (for blocks/SVM)
DBZ	Downline Block Size
DEL	Delete character
DLFP	Debug Log File Postprocessor utility
DN	Destination Node
DSR	Data Set Ready
DT	Device Type
EBCDIC	Extended Binary Coded Decimal Interchange Code
EC	Error Code
EOF	End of File
EOI	End of Information
EOJ	End of Job
EOM	End of Message
EOR	End of Record
EOT	End of Transmission
ETB	End of Transmission Block
ETX	End of Text
FD	Forward Data (block protocol)
FDX	Full Duplex
FE	Format Effector
FET	File Environment Table
FF	Form Feed
FN	Field Number
FS	Forward Supervision (block protocol)
FV	Field Value
HA	Header Area
HASP	Houston Automatic Spooling Program Protocol
HDLC	High-level Data Link Control
HDX	Half Duplex
HIP	Host Interface Package
HO	Host Ordinal
HOP	Host Operator
IAF	Interactive Facility program
ICMD	Interrupt Command
ICMDR	Interrupt Command Response
ICT	Input Character Type
INITN	Initialization Block Acknowledgment
INITR	Initialization Block Request
ISO	International Standards Organization
IVT	Interactive Virtual Terminal
LCF	Local Configuration File
LF	Line Feed
LFG	Load File Generator
LIP	Link Interface Package
LP	Line printer
MCS	Message Control System
MLIA	Multiplex Loop Interface Adapter
MPLINK	The Pascal link editor
MSG	End-of-message block
MTI	Message Type Indicators (Mode 4 protocol)
NAK	Negative Acknowledgment Block
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	Network Operator
NPU	Network Processing Unit
NS	Network Supervisor program
NVF	Network Validation Facility
ODD	Output Data Demand (Multiplex subsystem)
PA	Parity
PAD	Packet Assembly/Disassembly
PDN	Public Data Network
PFC	Primary Function Code
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
RCB	Record Control Byte (HASP protocol)
ROM	Read Only Memory
RR	Receive Ready (trunk or X.25 protocol)
RS	Reverse Supervision (block protocol)
RST	Reset Block
RTS	Request to Send
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
S-Frame	Supervisory Frame (trunk or X.25 protocol)
SRCB	Subrecord Control Byte (HASP protocol)
STX	Start of Text
SVM	Service Module (for processing service messages)
SYNC	Synchronizing Element
TAA	Text Area Array
TAF	Transaction Facility
TC	Terminal Class
TCB	Terminal Control Block
TIP	Terminal Interface Program
TLC	Text Length in Characters
TLMAX	Text Length Maximum
TNAME	Terminal Name
TO	Timeout
TTY	Teletypewriter
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
XBZ	Transmission Block Size
X-OFF	Stop character (ASYNC protocol)
X-ON	Start character (ASYNC protocol)
XPT	Transparent
X.3	CCITT protocol for asynchronous terminal access to a packet-switching network
X.25	CCITT protocol for packet-switching networks
X.28	CCITT protocol for terminal access to PSN/PAD
X.29	CCITT protocol for host access to PSN/PAD

APPLICATION 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	5-16
CALL NETDBG(dbugsup,dbugdat,avail)	6-7
CALL NETDMB(dumpid,ecs)	6-9
CALL NETGET(acn,ha,ta,tlmax)	5-4
CALL NETGETF(acn,ha,na,taa)	5-6
CALL NETGETL(aln,ha,ta,tlmax)	5-10
CALL NETGTFL(aln,ha,na,taa)	5-12
CALL NETLGS(address,size)	6-15
CALL NETLOG(address,size,format)	6-9
CALL NETOFF	5-4
CALL NETON(aname,nsup,status,minacn, maxacn)	5-1
CALL NETPUT(ha,ta)	5-7
CALL NETPUTF(ha,na,taa)	5-8
CALL NETREL(lfn,msglth,nrewind)	6-7
CALL NETSETF(flush,fetadr)	6-8
CALL NETSETP(option)	5-15
CALL NETSTC(onoff,avail)	6-15
CALL NETWAIT(time,flag)	5-14
CALL NSTORE(array,field,value)	4-11
[ivalue=]NFETCH(array,field)	4-12
ENTER FORTRAN-X QTCLOSE	8-15
ENTER FORTRAN-X QTENDT	8-14
ENTER FORTRAN-X QTGET USING ta-in	8-13
ENTER FORTRAN-X QTLINK	8-14
ENTER FORTRAN-X QTOPEN USING net-info-table	8-10
ENTER FORTRAN-X QTPUT USING ta-out-acn$_i$	8-11
ENTER FORTRAN-X QTTIP USING ta-out-acn$_i$	8-14

ASSEMBLY LANGUAGE LEVEL (NETTEXT-RESIDENT)

Call Format		Page
[label] NETCHEK		5-16
[label] NETDBG	dbugsup,dbugdat, avail	6-7
label2 NETDBG	dbugsup,dbugdat, avail,LIST	6-7
[label1] NETDBG	{LIST=label2 / LIST=register name}	6-7
[label] NETDMB	dumpid,ecs	6-9
label2 NETDMB	dumpid,ecs,LIST	6-9
[label1] NETDMB	{LIST=label2 / LIST=register name}	6-9
[label] NETGET	acn,ha,ta,tlmax	5-4
label2 NETGET	acn,ha,ta,tlmax, LIST	5-4
[label1] NETGET	{LIST=label2 / LIST=register name}	5-4
[label] NETGETF	acn,ha,na,taa	5-6
label2 NETGETF	acn,ha,na,taa,LIST	5-6
[label1] NETGETF	{LIST=label2 / LIST=register name}	5-6
[label] NETGETL	aln,ha,ta,tlmax	5-10
label2 NETGETL	aln,ha,ta,tlmax,LIST	5-10
[label1] NETGETL	{LIST=label2 / LIST=register name}	5-10
[label] NETGTFL	aln,ha,na,taa	5-12
label2 NETGTFL	aln,ha,na,taa,LIST	5-12

Call Format		Page
[label1] NETGTFL	{LIST=label2 LIST=register name}	5-12
[label] NETLGS	address,size	6-15
label2 NETLGS	address,size,LIST	6-15
[label1] NETLGS	{LIST=label2 LIST=register name}	6-15
[label] NETLOG	address,size,format	6-9
label2 NETLOG	address,size,format, LIST	6-9
[label1] NETLOG	{LIST=label2 LIST=register name}	6-9
[label] NETOFF		5-4
[label] NETON	aname,nsup,status, minacn,maxacn	5-1
label2 NETON	aname,nsup,status, minacn,maxacn,LIST	5-1
[label1] NETON	{LIST=label2 LIST=register name}	5-1
[label] NETPUT	ha,ta	5-7
label2 NETPUT	ha,ta,LIST	5-7
[label1] NETPUT	{LIST=label2 LIST=register name}	5-7
[label] NETPUTF	ha,na,taa	5-8
label2 NETPUTF	ha,na,taa,LIST	5-8
[label1] NETPUTF	{LIST=label2 LIST=register name}	5-8
[label] NETREL	lfn,msglth,nrewind	6-7
label2 NETREL	lfn,msglth, nrewind,LIST	6-7
[label1] NETREL	{LIST=label2 LIST=register name}	6-7
[label] NETSETF	flush,fetadr	6-8
label2 NETSETF	flush,fetadr,LIST	6-8
[label1] NETSETF	{LIST=label2 LIST=register name}	6-8
[label] NETSETP	option	5-15
label2 NETSETP	option,LIST	5-15
[label1] NETSETP	{LIST=label2 LIST=register name}	5-15
[label] NETSTC	onoff,avail	6-15
label2 NETSTC	onoff,avail,LIST	6-15
[label1] NETSTC	{LIST=label2 LIST=register name}	6-15
[label] NETWAIT	time,flag	5-15
label2 NETWAIT	time,flag,LIST	5-15
[label1] NETWAIT	{LIST=label2 LIST=register name}	5-15
[label] NFETCH	array,field, {Xj Bj}	4-10
[label] NSTORE	array,field=value	4-11

INDEX

COMMENT SHEET

MANUAL TITLE: Network Access Method Version 1
Host Application Programming Reference Manual

PUBLICATION NO.: 60499500

REVISION: W

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 on the back (please include page number references).

_____ Please reply   _____ No reply necessary

FOLD   FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 8241   MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Technology and Publications Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD   FOLD

CUT ALONG LINE

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE   TAPE